MDTF Getting Started Guide
Release 3.0 beta 2

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

Aug 01, 2020

Getting started

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.

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].

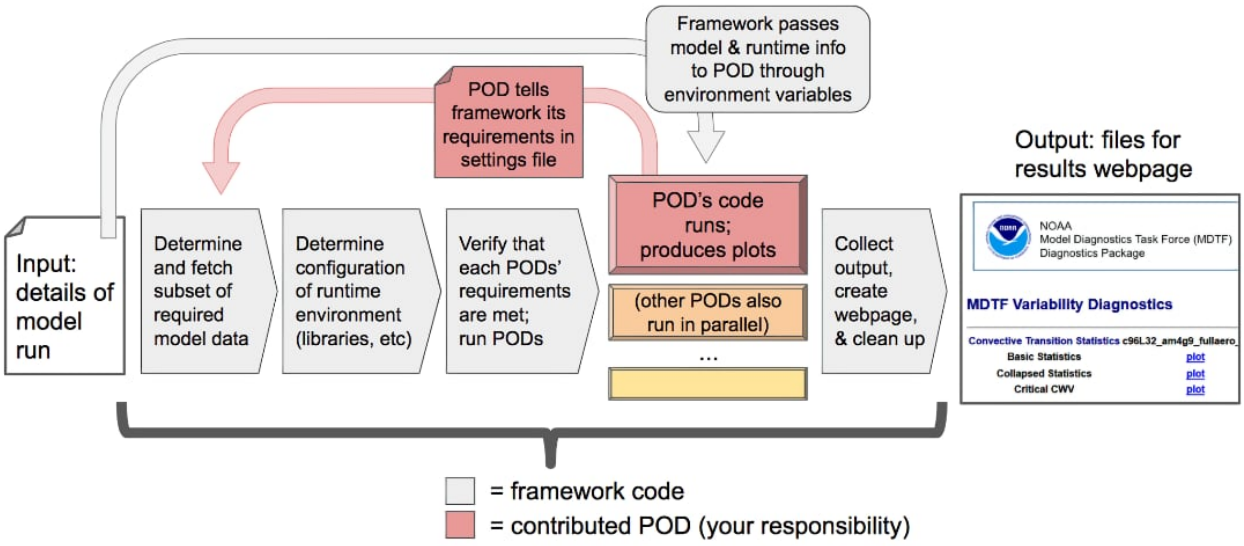
Also see the section of this site devoted to documentation of individual diagnostics.

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.

[H]

[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.)

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 some sample data.

We describe the most common configuration options (page 7) 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].

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.

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.

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

A checklist of items needed for submitting your POD for inclusion in the framework.

A collection of links to relevant tutorials and resources.

Quickstart installation instructions

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

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. While the package contains quite a few scripts, the most relevant for present purposes are:

`conda_env_setup.sh`: automated script for installing necessary Conda environments.

`default_tests.jsonc`: configuration file for running the framework.

Summary of steps for running the package

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

Download the package code and sample data for testing

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/`: 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 work on this branch as described in ref-dev-git.

---

https://github.com/NOAA-GFDL/MDTF-diagnostics
https://github.com/NOAA-GFDL/MDTF-diagnostics/releases/tag/v3.0-beta.1
https://github.com/NOAA-GFDL/MDTF-diagnostics/releases/tag/v3.0-beta.1

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 most relevant for package installation and default tests 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].

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
                   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 sample. The GFDL.CM4.c96L32.am4g10r8 sample 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 below in Section .

Install the necessary programming languages and modules

For users unfamiliar with Conda, :numref:`ref-conda-install` can be skipped if Conda has been installed, but :numref:`ref-conda-env-install` CANNOT be skipped regardless.

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 Miniconda[10] and Anaconda[11] python distribution, 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 provided script `src/conda/conda_env_setup.sh`, 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.

---

ftp://ftp.cgd.ucar.edu/archive/mdtf/MDTF_v2.1.a.obs_data.tar
ftp://ftp.cgd.ucar.edu/archive/mdtf/model.QBOi.EXP1.AMIP.001.tar
ftp://ftp.cgd.ucar.edu/archive/mdtf/model.GFDL.CM4.c96L32.am4g10r8.tar
https://docs.conda.io/en/latest/
https://docs.conda.io/en/latest/miniconda.html
https://www.anaconda.com/

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

Here we are checking that the Conda command is available on your system. We recommend doing this via Miniconda or Anaconda installation. You can proceed directly to section 2.2 if Conda is already installed.

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 the command doesn't return anything, i.e., you do not have a pre-existing Conda on your system, we recommend using the Miniconda installer available here[12]. Any version of Miniconda/Anaconda (2 or 3) released after June 2019 will work. Installation instructions here[13].

Toward the end of the installation process, enter "yes" at "Do you wish the installer to initialize Miniconda2 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`).

Restart the terminal to reload the updated shell login script.

Mac OS users may encounter a benign Java warning pop-up: To use the "java" command-line tool you need to install a JDK. It's safe to ignore it.

The framework's environments will co-exist with an existing Miniconda/Anaconda 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, e.g., environment modules.)

Framework-specific environment installation

Here we set up the necessary environments needed for running the framework and individual PODs via the provided script. These are sometimes referred to as "Conda environments" conventionally.

After making sure that Conda is available, 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 this path points to `/usr/` or a subdirectory therein, we recomnend having a separate Miniconda/Anaconda installation of your own following Conda installation (page 5).

Next, run

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

to install all necessary environments (and create an executable; Location of the MDTF executable (page 6)), 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 any other environments.

Substitute the actual paths for `$CODE_ROOT`, `$CONDA_ROOT`, and `$CONDA_ENV_DIR`.

The `--env_dir` flag allows you to put the program files in a designated location `$CONDA_ENV_DIR` (for space reasons, or if you don't have write access). You can omit this flag, and the environments will be installed within `$CONDA_ROOT/envs/` by default.

The `--all` flag makes the script install all environments prescribed by the YAML (.yml) files under `src/conda/` (one YAML for one environment). You can install the environments selectively by using the `--env` flag instead. For instance, `% ./src/conda/conda_env_setup.sh --env base --conda_root $CONDA_ROOT --env_dir $CONDA_ENV_DIR` will install the "_MDTF_base" environment prescribed by `env_base.yml`, and so on. With `--env`, the current script can install one environment at a time. Repeat the command for multiple environments.

Note that _MDTF_base is mandatory for the framework's operation, and the other environments are optional, see Framework interaction with Conda environments (page 7).

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. These

https://docs.conda.io/en/latest/miniconda.html
https://docs.conda.io/projects/conda/en/latest/user-guide/install/linux.html

environments can be uninstalled by simply deleting "_MDTF" directories under `$CONDA_ENV_DIR` (or `$CONDA_ROOT/envs/` for default setting).

Configure package paths

`src/default_tests.jsonc` is a template/example for configuration options that will be passed to the executable as an input. Open it in an editor (we recommend working on a copy). The following adjustments are necessary 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` pointing to `mdtf/inputdata/obs_data/` and `mdtf/inputdata/model/` 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 (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.

`conda_root` should be set to the value of `$CONDA_ROOT` used above in Framework-specific environment installation (page 5).

If you specified a custom environment location with `$CONDA_ENV_DIR`, set `conda_env_root` to that value; otherwise, leave it blank.

We recommend using absolute paths in `default_tests.jsonc`, but relative paths are also allowed and should be relative to `$CODE_ROOT`.

Run the MDTF package with default test settings

Location of the MDTF executable

The setup script (Framework-specific environment installation (page 5)) will have created an executable at `$CODE_ROOT/mdtf` which sets the correct Conda environments before running the framework and individual PODs. To test the installation, `% $CODE_ROOT/mdtf --help` will print help text on the command-line options. Note that, if your current working directory is `$CODE_ROOT`, you will need to run `% ./mdtf --help`.

For interested users, the `mdtf` executable is also a script, which calls `src/conda/conda_init.sh` and `src/mdtf.py`.

Run the framework on sample data

If you've installed the Conda environments using the `--all` flag (Framework-specific environment installation (page 5)), you can now run the framework on the CESM sample model data:

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

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

If you edited/renamed `default_tests.jsonc`, pass that file instead.

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

The above command will execute PODs included in `pod_list` of `default_tests.jsonc`. Skipping/adding certain PODs by uncommenting/commenting out the POD names (i.e., deleting/adding `//`). Note that entries in the list must be separated by `,` properly. Check for missing or surplus `,` if you encounter an error (e.g., "ValueError: No closing quotation").

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" of `default_tests.jsonc`, and uncomment the section for GFDL.CM4.c96L32.am4g10r8.

Framework interaction with Conda environments

As just described in Run the framework on sample data (page 6), when you run the `mdtf` executable, among other things, it reads `pod_list` in `default_tests.jsonc` and executes POD codes accordingly. For a POD included in the list (referred to as $POD_NAME):

1. The framework will first try to determine whether there is a Conda environment named `_MDTF_$POD_NAME` under `$CONDA_ENV_DIR`. If yes, the framework will switch to this environment and run the POD.
2. If not, the framework will then look into the POD's `settings.jsonc` file in `$CODE_ROOT/diagnostics/$POD_NAME`. `runtime_requirements` in `settings.jsonc` specifies the programming language(s) adopted by the POD:

   a). If purely Python, the framework will switch to `_MDTF_python3_base` and run the POD (_MDTF_python2_base for ealier PODs developed in Python 2.7).

   b). If NCL is used, then `_MDTF_NCL_base`.

   If you choose to selectively install Conda environments using the `--env` flag (Framework-specific environment installation (page 5)), remember to install all the environments needed for the PODs you're interested in, and that `_MDTF_base` is mandatory for the framework's operation.

   For instance, the minimal installation for running the `EOF_500hPa` and `convective_transition_diag PODs` requres `_MDTF_base` (mandatory), `_MDTF_NCL_base` (because of b), and `_MDTF_convective_transition_diag` (because of 1). These can be installed by passing `base`, `NCL_base`, and `convective_transition_diag` to the `--env` flag one at a time (Framework-specific environment installation (page 5)).

Next steps

Consult the next section (page 7) for how to run the framework on your own data and configure general settings.

Using customized model data & framework configuration

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[14]. 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 only set the 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 using customized model data

1. Have your data files ready in NetCDF format.
2. Save the files following the specified directory hierarchy and filename convention.
3. Check the variable name convention.
4. Edit the configuration input file accordingly, then run the framework.

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

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. 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, you can make symlinks[15] 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)
      GFDL.CM4.c96L32.am4g10r8
      QBOi.EXP1.AMIP.001
      my_new_experiment
          day
              my_new_experiment.pr.day.nc
```

As implicitly demonstrated by the examples above, the current framework does not support having multiple variables in one file.

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

Check variable name convention

The variable names have to be consistent with the convention expected by the framework, as defined in the JSON files `src/fieldlist_$convention.jsonc`.

Currently, the convention templates provided by the framework include `CMIP`, for CF-compliant output produced as part of CMIP6 (e.g.,, by post-processing with CMOR[16]) and `CESM`, `AM4` and `SPEAR`. We hope to offer support for the native variable naming conventions of a wider range of models in the future.

For instance, `src/fieldlist_CESM.jsonc` specifies the convention adopted by the NCAR CESM. Open this file, the line `"pr_var" : "PRECT",` means that total precipitation rate (pr for CF-compliant output) should be saved as PRECT (case sensitive). In addition, `"pr_conversion_factor" : 1000,` makes the units of precipitation CF-compliant.

You can either change the NetCDF variable/file names following the provided `fieldlist_$convention.jsonc` files, or edit and rename the files to fit your model data.

Note that entries in the JSON files must be properly separated by `,`. Check for missing or surplus , if you encounter an error

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[17] 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. With the string specified here (referred to as $convention), the framework will look for the corresponding `src/fieldlist_$convention.jsonc`

`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 (e.g.,, 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 (see Diagnostics reference[18] for variables required by each POD). 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.

Other framework settings

The paths to input and output data described in Configure package paths (page 6) 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 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.

---

https://cmor.llnl.gov/
https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/develop/src/default_tests.jsonc
https://mdtf-diagnostics.readthedocs.io/en/latest/sphinx/pod_toc.html

Modifying POD settings

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

Troubleshooting

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

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[19] 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`.

Issues with standalone NCL installation

Many Linux distributions (Ubuntu, Mint, etc.) have offered a way of installing NCL[20] 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
```

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.

Site-specific information

[19] https://imagemagick.org/index.php
[20] https://www.ncl.ucar.edu/

GFDL-specific information

This page contains information specific to the site installation at the Geophysical Fluid Dynamics Laboratory[21].

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.

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[22] 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`.

2. As a batch job on PPAN, managed via slurm. This is handled via the mdtf_gfdl_interactive.csh[23] wrapper script.

3. Called from an interactive shell on PPAN or workstations.

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.

---

https://www.gfdl.noaa.gov/
https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl.csh
https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl_interactive.csh

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[24] 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[25] 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.

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/0ar.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.

Acknowledgements Development of this code framework for process-oriented diagnostics was supported by the National Oceanic and Atmospheric Administration[26] (NOAA) Climate Program Office Modeling, Analysis, Predictions and Projections[27] (MAPP) Program (grant # NA18OAR4310280). Additional support was provided by University of California Los Angeles[28], the Geophysical Fluid Dynamics Laboratory[29], the National Center for Atmospheric Research[30], Colorado State University[31], Lawrence Livermore National Laboratory[32] and the US Department of Energy[33].

Many of the process-oriented diagnostics modules (PODs) were contributed by members of the NOAA Model Diagnostics Task Force[34] under MAPP support. Statements, findings or recommendations in these documents do not necessarily reflect the views of NOAA or the US Department of Commerce.

---

https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl.csh
https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl.csh
https://www.noaa.gov/
https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP
https://www.ucla.edu/
https://www.gfdl.noaa.gov/
https://ncar.ucar.edu/
https://www.colostate.edu/
https://www.llnl.gov/
https://www.energy.gov/
https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP/MAPP-Task-Forces/Model-Diagnostics-Task-Force

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.