v.0 document: Getting_started_v2.0.pdf

Getting started with the MDTF package

Yi-Hung Kuo (UCLA), Dani Coleman (NCAR).
Last update: XXX

This document provides directions for downloading, installing and running a test of the Model Diagnostics Task Force (MDTF) Process-Oriented Diagnostics package using sample model data.

For descriptions of the Process-Oriented Diagnostic modules (referred to as PODs hereafter) included in the package, and sample output, visit the MDTF main page. The package is provided as-is. Questions about individual PODs should go to the contributing groups.

For developers contributing a POD to the package, see Developers' Walkthrough.

Summary of steps for running the package:

I. Download the source code(a) code package, (b) digested observational data, and (c) two sets of sample model data. See section 1.
II. Install (a) NCL and (b) Python with libraries. See section 2.
III. Run the package by executing python mdtf.py namelist in a terminal. See section 3.

Below, sections 1-4 summarize steps for testing the package with provided samples. For advanced users, sections 5-6 detail how to run the package with your own model data. Section 7 is a very short list for troubleshooting.

1. Download the package code and sample data for testing

1.1 Obtaining the code

Change from v2.0: source code is distributed via github.

The MDTF code is distributed through the GitHub repository at github.com/NOAA-GFDL/MDTF-diagnostics.

- Official releases are available by selecting "releases" from this screen. Code can be downloaded as .zip or .tar.gz files from the links at the end of the release description. The most recent release is recommended for users of the package.
- The "master" branch contains features that haven't yet been incorporated into an official release, which may be less stable or thoroughly tested. This can be downloaded as a

.zip file or git-cloned using the green button on the github page. This option is recommended for users interested in keeping more up-to-date on project development and contributing feedback.
- The “develop” branch is the “beta test” version of the framework. It may be selected from the “Branch: master” button on the main page, and then downloading or git-cloning. POD developers should begin work on this branch as described in the Developer’s Walkthrough.

Installing the code will create a directory titled “MDTF-diagnostics” containing the files listed on the github page. Below we refer to this MDTF-diagnostics directory as $CODE_ROOT. It contains the following subdirectories:

- diagnostics/ : directories containing source code of individual PODs
- doc/ : directory containing documentation (a local mirror of the github wiki and documentation site)
- src/ : source code of the framework itself
- tests/ : unit tests for the framework

1.2 Obtaining supporting data

Supporting observational data and sample model data is available via anonymous FTP at ftp://ftp.cgd.ucar.edu/archive/mdtf. The observational data is required for the PODs’ operation; the sample model data is provided for test/demonstration purposes and is optional.

Note these tar files were created using settings incompatible with BSD tar, so users installing on Mac OS should use the finder’s Archive Utility.app instead of the command-line tar command to extract the files.

- Digested observational data: ftp://ftp.cgd.ucar.edu/archive/mdtf/MDTF_v2.1.a.var_code.tar
- Sample model data: ftp://ftp.cgd.ucar.edu/archive/mdtf/model.QBOi.EXP1.AMIP.001.tar
- Additional sample data: ftp://ftp.cgd.ucar.edu/archive/mdtf/model.GFDL.CM4.c96L32.am4g10r8.tar

If it’s not done by the extraction command, place the contents of these archives in the following hierarchy under a directory called “mdtf” [1] :

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

[1] Throughout this document, % indicates the UNIX/LINUX command line prompt and is followed by command to be executed in a terminal, and $ indicates strings to be substituted, e.g., the string $ver here should be substituted by the actual version in the tar file name.

Here, the main script mdtf.py will call POD scripts under var_code following the settings specified in namelist. The observational data is stored under obs_data. The sample model data, one from a particular run of the NCAR CESM (CASENAME: QBOi.EXP1.AMIP.001), and another from a GFDL CM4 run (CASENAME: GFDL.CM4.c96L32.am4g10r8), are under model.

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, see section 3. For instructions on using your own data set, see section 5.

2. Install the necessary programming languages and modules

Changes from v2.0: conda recommended to install all programs; conda environments provided for each POD; setup is handled by an automated script.

The MDTF framework code is written in Python 2.7, but supports running PODs written in a variety of scripting languages. To handle this, we recommend that all users manage these languages and their libraries through the conda package manager, even if they have independent installs of these languages on their machine.

Note that the framework only makes use of the conda package manager, which is free, open source software. The conda package manager is one component of the Anaconda python distribution; having Anaconda is sufficient but not necessary/required to obtain conda.

2.1 Conda installation

The framework's environments can co-exist within an existing conda or Anaconda installation. Because conda/anaconda sets paths in the user's shell login scripts, installing multiple copies of conda/Anaconda will necessarily break one or the other (note this is distinct from having multiple environments managed by a single installation of conda, which is what we want). Run

```
% conda --version
```

as the user who will be using the framework to determine if conda is installed; the framework has been tested against versions of conda >= 4.7.5.

If you do not have a pre-existing conda installation on your system, we recommend using the "miniconda2" (python 2.7) installer from https://docs.conda.io/en/latest/miniconda.html, but any current version of miniconda or Anaconda will work.

2.2 Conda environment installation

You should decide if you want to install the conda environments used by the framework into a location separate from your other conda environments. Note by default, conda uses hard symbolic links instead of duplicating files, so the overall disk space usage is the same in either case (although it may be reported differently depending on your system).

The names of all framework-created environments begin with "_MDTF", so as not to conflict with any other environments that are defined. If a custom installation location is used, the environments can only be activated by their prefix (not their name), providing a further degree of isolation from your other conda environments.

To install environments in the standard/user location, run

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

To install in a custom location named $ENV_DIR (a recommended location is $CODE_ROOT/envs, which keeps the environments with the source code), instead run

```
% $CODE_ROOT/src/conda/conda_env_setup.sh --all --env_dir $ENV_DIR
```

The installation process can be time-consuming. After installing the framework-specific conda environments, you shouldn't manually alter them (ie, don't run "conda update" on them).

2.3 Non-conda installation

The framework can make use of existing language installations outside of conda environments; however, we support this mode of operation at a lower priority since it becomes dependent on the details of each user's system.

The following software is used by the framework:

- Python version 2.7: framework will attempt to create virtualenvs for each pod
- NCO utilities version 4.5.4
- ImageMagick
- NCL, version 6.5.0 or newer
- R (for the SM_ET_coupling POD only)

3. Execute the MDTF package

Changes from v2.0: location of executable changed; names of paths and configuration mechanism changed.

3.1 Location of the MDTF executable

If you opted for the conda-based installation, the conda_env_setup.sh script will have created a wrapper script at $CODE_ROOT/mdtf which sets the correct conda environment before running the framework. To test the installation,

% $CODE_ROOT/mdtf --help

should print help on the command-line options.

If you didn't install with conda, the framework is instead started from the python script at $CODE_ROOT/src/mdtf.py. Replace "$CODE_ROOT/mdtf" in what follows with "$CODE_ROOT/src/mdtf.py".

3.2 Configuring installation paths

The framework needs the paths to relevant data and programs. There are two ways to pass this information; in both cases, shell variables beginning with '$' will be expanded at run time, so you can specify, eg, "$HOME/dir_name" or "$TMPDIR". As an example, we'll show how to set the value of OUTPUT_DIR to "$HOME/mdtf_out".

- As long-form command-line options; you'd run the framework as
  % $CODE_ROOT/mdtf --OUTPUT_DIR $HOME/dir_name [... other options...]
- Editing the configuration defaults. These are set in a commented JSON file in $CODE_ROOT/src/defaults.jsonc (which define the command-line options.)
  - If you didn't clone the git repo, you can edit this file directly. In the example, you'd find the entry containing "name": "OUTPUT_DIR", and change the line in that entry beginning with "default" to read "default": "$HOME/dir_name". Other lines in the entry shouldn't be changed.
  - If you cloned the git repo, you will have to be careful to not commit the changes to this file. A less fragile way to do this is to make the above changes to a copy of $CODE_ROOT/src/defaults.jsonc, and specify this when running the framework as
    % $CODE_ROOT/mdtf -f my_defaults.jsonc [... other options...]
- Options set explicitly on the command line always override those set in a copied defaults file (set with -f above), if present, and options in that file override those in the framework's defaults.jsonc.

Paths that you should consider configuring are:

- MODEL_DATA_ROOT: path to the sample model data downloaded in step 1.2. If you didn't rename directories, this would end in ".../inputdata/model".
- OBS_DATA_ROOT: path to the supporting observational data downloaded in step 1.2. If you didn't rename directories, this would end in ".../inputdata/obs_data".

- WORKING_DIR: Parent directory the framework will use for downloading remotely-stored data (when not using the sample data) and collecting POD output.
- OUTPUT_DIR: Parent directory that the framework will copy finished results to. By default, results of different runs are given unique names so that your data is never overwritten.
- conda_root: Only used in conda-based installations. Location of your conda installation. The framework will attempt to determine this automatically if not specified, but due to technical issues with conda it's recommended you set this manually. Set this path to the value returned by running
  % conda info --base
- conda_env_root: Only used in conda-based installations. If you chose to install the MDTF conda environments in a custom location in step 2.2, this should be set to the value of $ENV_DIR you used. If you didn't use a custom location, this should be set to the empty string "".
- venv_root: Only used in non-conda-based installations. Directory where the framework should create python virtualenvs.
- r_lib_root: Only used in non-conda-based installations. Directory where the framework should install user R libraries.

3.3 Run the framework on sample data

By default, the framework will run on the QBOi.EXP1.AMIP.001 run of the sample model data. This is specified in the "caselist" section of defaults.jsonc, and may be changed by passing different values for those attributes on the command line (see next section).

To run the framework on this sample data, execute
% $CODE_ROOT/mdtf <path to QBOi.EXP1.AMIP.001>

Output will be in a directory titled QBOi.EXP1.AMIP.001_1977_1981 in the directory specified as OUTPUT_DIR. Open index.html in a web browser to view the output. Results should match this sample webpage output of the package.

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

4. Using your own model data

As of this writing, the framework only supports running on model data on a local filesystem in a pre-set directory structure.

4.1 Check your model's variable name convention.

The framework is capable of translating variable names from the conventions used by various models; however, at the moment native support for different models is limited to CAM/CESM2 (NCAR) and AM4 (GFDL).

The framework also supports the CF conventions as used in CMIP6, so the currently supported mode of operation is to run your model's output through CMOR to ensure variable names and units are CMIP6-compliant.

4.2 Place model data in the expected directory structure.

This should follow the structure used for the sample model data. Subdirectories should be created according to data frequency, and data files should be renamed as follows:

$CASENAME/
6hr/, 3hr/, 1hr/, day/, mon/
$CASENAME.<variable name>.<data_frequency>.nc

For example, QBOi.EXP1.AMIP.001/day/QBOi.EXP1.AMIP.001.PRECT.day.nc. Below we refer to the absolute path to this directory (ie, .../$CASENAME) as $CASE_DIR.

Symbolic links may be used in place of copies of data files if your model data is stored in a different directory structure.

4.3 Framework invocation

See the complete list of command-line options at https://github.com/NOAA-GFDL/MDTF-diagnostics/wiki/Command-line-reference.

% $CODE_ROOT/mdtf --CASENAME $CASENAME --FIRSTYR <starting analysis year> --LASTYR <final analysis year> --OUTPUT_DIR <output location> $CASE_DIR.

The framework also recognizes shortened versions of these flags:
% $CODE_ROOT/mdtf -n $CASENAME -Y <starting analysis year> -Z <final analysis year> -o <output location> $CASE_DIR.

For repeated analysis, the above parameters can also be input via the "caselist" section of the defaults configuration file.

Log information will be printed to the terminal. Output will be saved in a directory titled $CASENAME_$FIRSTYR_$LASTYR in the specified location. Open the index.html file in that directory to view the results.

5. Modifying package settings

Changes from v2.0: The “defaults.jsonc” file and the command-line options it defines replace the previous namelist file format.

See section 3.2 for how to set command-line flags, and optionally how to change their default values by modifying the src/default.jsonc file. In addition to the paths listed in that section, other useful settings are:

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

As with the paths, these can be set as command-line flags each time the framework is run, or (if not present on the command line) according to their default values in defaults.jsonc.

6. Troubleshooting

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

1) The error message “convert: not authorized …” shows up:

The MDTF package generates figures in the PostScript (PS) format, and then uses the convert command (from the ImageMagick 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 [2], change line
<policy domain=”coder” rights=”none” pattern=”PS” />
to
<policy domain=”coder” rights=”read|write” pattern=”PS” />

[2] The folder name ImageMagick may depend on its version, e.g., ImageMagick-6.

2) Issues with standalone NCL installation

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

3) Issues with the convective transition POD:

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

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/ [3]

Manually copying the object file to ~/anaconda2/lib/ should solve the error.

[3] ~/anaconda2/ is where Anaconda 2 is installed. The precise names of the object file and openssl-folder may vary.