Developers' walkthrough for the MDTF Diagnostic Framework v2.0: an example using the Convective Transition Diagnostic module

Yi-Hung Kuo,[a] Dani Coleman,[b] Chih-Chieh Jack Chen,[b] Andrew Gettelman,[b] J. David Neelin,[a] Eric Maloney.[c]
(a: UCLA; b: NCAR; c: CSU)
Last update: 3/11/2019

This walkthrough contains a brief overview on the structure of the framework, with emphasis on the pseudo-code—structural level—rather than on the implementation details, although tips on implementation and testing are included. The purpose of this walkthrough is to provide guidelines facilitating the development of the package toward the end phase when modules by the participating groups are integrated into the framework. This walkthrough accompanies a code package including example modules. Visit the MDTF main page for more about this framework.

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 coordinating team, and the *PODs* (or *modules*) are developed by individual groups (or *developers*). PODs are independent of each other. Each POD: (1) will produce its own html file (webpage) as the final product, (2) consists of a set of process-oriented diagnostics, and (3) each such diagnostic will produce figure(s) that can be displayed by the html file in a browser.

For the rest of this walkthrough, the Convective Transition Diagnostic POD developed by the UCLA group is used as a concrete example to illustrate how a POD is implemented and integrated into the framework. We assume that the POD developers (1) have developed their own set of working scripts that read in model output and pre-digested observations and produce figures displaying the diagnostic analysis results, and (2) have successfully tested the MDTF code package following the Getting-Started document.

As the developer learned in the Getting Started document, the package directory structure is:

mdtf/MDTF_v2.0[1]	*(DIAG_HOME* containing the *mdtf.py* script*)*
mdtf/MDTF_v2.0/var_code	*(VARCODE* containing directories for each POD's source code*)*
mdtf/inputdata/obs_data	*(VARDATA* containing directories for each POD's observational data*)*
mdtf/inputdata/model	*(DATADIR)*

DATADIR contains model output for the *QBOi.EXP1.AMIP.001*[2] sample with the following sub-directories:

DATADIR/mon
DATADIR/day
DATADIR/6hr
DATADIR/3hr
DATADIR/1hr

[1] Directories/files/scripts/codes are in *Italics*. Here, *DIAG_HOME*, *VARCODE*, *VARDATA*, and *DATADIR* are environment variables. All environment variables are listed in Appendix 1 at the end of this walkthrough.

[2] Here *QBOi.EXP1.AMIP.001* is the case name of the sample model run to be analyzed, set by the *CASE* string in the text file given as input to *mdtf.py* (by default, *namelist*). Python scripts later called by *mdtf.py* can access this variable through *os.environ["CASENAME"]*.

1. Get started with the code package

The code package is available at the MDTF main page (see Downloading and Running), along with the pre-digested observations and sample model data. The POD developers should test the code package following instructions in the Getting-Started document to ensure they understand the structure and have the required environment.

2. To-do list

The following are the necessary 6 steps for the module implementation and integration into the framework. Here the POD name tag is *convective_transition_diag*.[3] It is to be replicated with different POD name tags. All the modules currently included in the code package have the same structure, and hence the descriptions below apply:

1. Provide all the scripts for the *convective_transition_diag* POD in the sub-directory *DIAG_HOME/var_code/convective_transition_diag*. Among the provided scripts, there should be a template html file *convective_transition_diag.html*,[4] and a main script *convective_transition_diag.py* that calls the other scripts in the same sub-directory for analyzing, plotting, and finalizing html.

2. Provide all the pre-digested observation data/figures in the sub-directory *DATA_IN/obs_data/convective_transition_diag*.

3. Add a line of code *POD convective_transition_diag* to the POD section of the *namelist* file (in *DIAG_HOME*) so that the *mdtf.py* script will call the *convective_transition_diag.py* script.

4. A section of code should be added to the end of the script *convective_transition_diag.py*[5] for copying and modifying the template *convective_transition_diag.html*, inserting a line into *index.html*, and, if necessary, converting figures in postscript format into PNG.

5. Provide documentation following the templates:

    1) Provide a comprehensive POD documentation, including a one-paragraph synopsis of the POD, developers' contact information, required programming language and libraries, and model output variables, a brief summary of the presented diagnostics as well as references in which more in-depth discussions can be found (see an example).

[3] The *POD name tag* used in the code should closely resemble the full POD name but should not contain any space bar or special characters. Note that the *convective_transition_diag* tag here is used repeatedly and consistently for the names of sub-directories, script, and html template. Please follow this convention so that *mdtf.py* can automatically process through the PODs.

[4] One can create a new html template by simply copying and modifying the example templates in *DIAG_HOME/var_code/html/html_template_examples*. Note that scripts therein are exact replications of the html-related scripts in the example PODs, serving merely as a reference, and are not called by *mdtf.py*.

[5] An example can be found at the end of *convective_transition_diag.py* (or scripts for the other PODs). In the sub-directory *DIAG_HOME/var_code/html/html_template_examples*, the script *set_html.py* contains a copy of the last sections of code from some PODs.

2) All scripts should be self-documenting by including in-line comments. The main script *convective_transition_diag.py* should contain a **comprehensive header** providing information that **contains the same items as in the POD documentation**, except for the "More about this diagnostic" section.

3) The **one-paragraph POD synopsis** (in the POD documentation) as well as a link to the **Full Documentation** should be placed at the top of the template *convective_transition_diag.html* (see example).

6. Test before distribution. It is important that developers test their POD before sending it to the MDTF contact. Please take the time to go through the following procedures:

1) Test how the POD fails. Does it stop with clear errors if it doesn't find the files it needs? How about if the dates requested are not presented in the model data? Can developers run it on data from another model? If it fails, does it stop the whole *mdtf.py* script? (It should contain an error-handling mechanism so the main script can continue). Have developers added any code to *mdtf.py*? (**Do not change *mdtf.py*!** — if you find some circumstance where it is essential, it should only be done in consultation with the MDTF contact).

2) Make a clean tar file. For distribution, a tar file with *obs_data/*, *var_code/*, *namelist*, and model data that developers have thoroughly tested is needed. These should not include any extraneous files (output NetCDF, output figures, backups, pyc, *~, or # files). The model data used to test (if different from what is provided by the MDTF page) will need to be in its own tar file. Use *tar -tf* to see what is in the tar file. Developers might find it helpful to consult the script used to make the overall distributions *mdtf/make_tars.sh*.

3) Final testing: Once a tar file is made, please test it in a clean location where developers haven't run it before. If it fails, repeat steps 1)-3) until it passes. Next, ask a colleague or assign a group member not involved in the development to test it as well — download to a new machine to install, run, and ask for comments on whether they can understand the documentation.

4) Post on an ftp site and/or email the MDTF contact.

3. The workflow

When *mdtf.py* is executed, among other things, it creates the following new sub-directories:

A. DIAG_HOME/wkdir
B. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001[6]

In sub-directory (B), each POD will create its own sub-directory, named after the POD, under which the html and figures produced by the POD will be saved (see subsection I below).

[6] In general, *DIAG_HOME/wkdir/MDTF_CASENAME*. This sub-directory is set to be the environment variable *variab_dir* in *mdtf.py* (and *WKDIR* = *DIAG_HOME/wkdir*).

The script *mdtf.py* then creates the file *index.html* in sub-directory (B) through which the html files created by the individual PODs are linked. This *index.html* file (also a script) will be subsequently modified by the PODs called by *mdtf.py* through the code added in step 4 (see subsection II below).

I. *mdtf.py* calls *convective_transition_diag.py* (i.e. the POD)

Through the *POD line* added to *namelist* in step 3, *mdtf.py* then calls *convective_transition_diag.py*. The latter script creates the following new sub-directories:

C. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001/convective_transition_diag
D. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001/convective_transition_diag/model
E. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001/convective_transition_diag/obs

If developers chooses to save the analyzed model or obs results (not including figures; referred to as the *intermediate output*) produced by the POD for later analysis or plotting, the following 2 additional sub-directories will be created by *convective_transition_diag.py*:

F. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001/convective_transition_diag/model/netCDF
G. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001/convective_transition_diag/obs/netCDF

in which the intermediate output (netCDF format recommended), for model and obs will be saved respectively. Similarly, if developers chooses to save the figures in postscript (.ps) format for later publication, the following 2 additional sub-directories will be created in which the postscript figures will be saved:

H. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001/convective_transition_diag/model/PS
I. DIAG_HOME/wkdir/MDTF_QBOi.EXP1.AMIP.001/convective_transition_diag/obs/PS

If a group chooses not to produce intermediate output or postscript figures, sub-directories (F)-(I) can be skipped.

Next, *convective_transition_diag.py* analyzes the model output and plots figures, by calling scripts provided in step 1, and saves the intermediate output (if any) for model in sub-directory (F). The corresponding model figures are saved in sub-directory (D). The same script also plots obs figures by reading data provided in step 2 and saves them in sub-directory (E). [7] If the obs figures are already provided in step 2, the script can simply copy the figures to the sub-directory.

II. *convective_transition_diag.py* finalizes *convective_transition_diag.html*

The last section of the script *convective_transition_diag.py* mentioned in step 4 copies the html template *convective_transition_diag.html* (step 1) to sub-directory (C), and then modifies the copied html template if the filenames of the figures just made and saved in sub-directories (D) and (E) depend on some parameters (e.g., *CASENAME*) and hence are different from what were in the original template. A link to *convective_transition_diag.html* is inserted to *index.html* through the same section of code (using the *echo* command). If the figures are saved as postscripts (.ps) in sub-directories (H) and (I), this section also converts the postscripts into PNG format (.png) and save the PNG figures for model and obs

[7] Apparently, postscript figures will be saved in sub-directories (H) and (I).

respectively in sub-directories (D) and (E). When the environment variable *CLEAN* is set to *"1"* in *namelist*,[8] sub-directories (H) and (I) together with their contents will then be removed.

An example of how this section is coded can be found at the end of *convective_transition_diag.py* (or the corresponding Python scripts for the other example PODs). Note that the html templates for some PODs can be found in *DIAG_HOME/var_code/html/html_template_examples*, and the last sections of code from these PODs are copied and saved as *set_html.py* therein, as a reference. The UCLA group, though had no prior knowledge of the html syntax, was able to copy and modify the examples provided by NCAR.

III. Back to *mdtf.py*

The last section of *mdtf.py* creates a tar file of sub-directory (B) as the final product of the code package. Note that postscript figures and intermediate output in netCDF format (if any) will not be included in the tar file.

IV. Summary

Subsections 3.I-III describe how the control is passed from *mdtf.py* to *convective_transition_diag.py* and back to *mdtf.py*, and what the POD scripts are expected to do. The developers should closely follow the example given here (e.g., the sub-directory structure). While developers can re-organize the structure and change names of the sub-directories (D)-(I) depending on how they choose to present their diagnostics, **sub-directory (C) is mandatory**, and **POD scripts should never create/modify directories or files outside sub-directory (C)**.

4. Required environment & A remark on installing NCL on a Linux machine

The current code package is developed and has been tested in Python 2.7. We recommend developers to use *Anaconda* for the Python environment management. See the Getting-Started document for more about this.

Note that Python 2 is planned to phase out by 2020, and the next iteration of the code package will transition to Python 3. So, having the POD scripts be compatible with Python 3 would be much desirable.

The rest of this section regards a known issue of NCL installation on a Linux machine. The NCAR Command Language (NCL) is required for some example PODs. In recent years, many Linux distributions (e.g., Ubuntu 16.04, Mint 18, etc.) have offered an easy way of installing NCL through *Synaptic Package Manager* or running the following command in a terminal (not recommended):

  sudo apt install ncl[9]

With NCL being installed this way, the users may encounter errors when running the example PODs provided by NCAR, even if the environment variables and search path have been added.[10] Installing NCL through Anaconda would not resolve this type of error either.

[8] Default *VAR CLEAN 0* in *namelist*.

[9] The command *apt* is gradually replacing the traditional *apt-get* for Debian-based distributions, but both would work. A Red Hat-based distribution (e.g., Fedora, CentOS) uses *dnf* instead.

If one encounter such errors, an easier solution is to install NCL by downloading the pre-compiled binaries following instructions in Getting-Started document (see section 2 therein).[11]

5. Model output variable and filename convention

There are 4 environment variables assigned through the *CASE* line in *namelist*: *CASENAME*, *model*, *FIRSTYR*, and *LASTYR*. For example, the CASE line for the sample *QBOi.EXP1.AMIP.001* case reads

CASE QBOi.EXP1.AMIP.001 CESM 1977 1981

Here, *FIRSTYR*, and *LASTYR* specify the time period for analysis, *CASENAME* points to locations of the model data (*DATADIR = DATA_IN/CASENAME*) and where the diagnostic analysis results are saved (*variab_dir = WKDIR/MDTF_CASENAME*). Most example PODs also display the *CASENAME* in figures to distinguish the results from observations or some reference model results.

Note that models developed by different centers adopt different model output variable naming conventions, e.g., precipitation rate is *pr/PRECT/pr* (case sensitive) in the CMIP/CESM/AM4 convention. The environment variable *model* is used exclusively for specifying the naming convention by pointing to the script *set_variables_model.py* under D*IAG_HOME/var_code/util*.[12]

The current version of the code package assumes the following filename structure for the model data files:

CASENAME.VARNAME.FREQ.nc

For instance, the 3-hourly and daily precipitation rate data for the *QBOi.EXP1.AMIP.001* sample have the following filenames, respectively:

QBOi.EXP1.AMIP.001.PRECT.3hr.nc[13]
QBOi.EXP1.AMIP.001.PRECT.day.nc

These files are stored in separate sub-directories according to their time frequency as described in the beginning of this walkthrough.

To summarize, in the filename structure, *CASENAME* is set in *namelist*, *VARNAME* (variable name) follows the naming convention determined by *model*. As for *FREQ* (time frequency), developers should specify the preferred time frequency for their PODs, or implement a mechanism to check through the available frequencies, e.g., check for the preferred 3-hourly data; if unavailable, go for 6-hourly and so on.

[10] This is because the NCL scripts in the example modules would try to load the other NCL scripts in the sub-directory *$NCARG_ROOT/lib/ncarg/nclscripts/csm* that comes with the NCL installation.

[11] Remember to remove the NCL installation through *Synaptic* or *apt* before re-install.

[12] Currently, there are 3 *set_variable* scripts in the code package, for CMIP, CESM, and AM4. These scripts do not contain a complete list of model variables. Developers can add new variables to the existing list as needed or create new *set_variable* scripts for other models, and submit the changes to the MDTF contact.

[13] Here the variable name for precipitation rate is set by *os.environ[“pr_var”] = “PRECT”* in *set_variables_CESM.py*.

6. Use environment variables

In the scripts *mdtf.py* and *set_variables_CESM.py*, many environment variables are defined through *setenv* or *os.environ*, and many environment variables can be overwritten by adding new *VAR* lines to *namelist*. These environment variables are to help make the PODs more robust. For instance, when the interested model data is not saved under the default location *DATADIR = DATA_IN/CASENAME*, it suffices to change the environment variable *DATADIR* pointing to where the model data are located by inserting the following line into *namelist*:

VAR DATADIR path_to_directory_containing_data

Appendix 1 at the end of this walkthrough provides a list of environment variables the POD scripts can access (e.g., using *os.environ* in Python or *getenv* in NCL). Developers can always overwrite the default values through *VAR* lines in *namelist*.[14] Please note that *mdtf.py* should never be changed without first consulting the MDTF contact.

A common mistake made by previous POD developers, as being noted by the MDTF coordinating team, is that the locations of POD scripts/model output/observational data, i.e., things that should be expressed in terms of the environment variables, are hardwired in the scripts. And when such PODs are submitted to the coordinating team, they cannot function as expected on a different machine and/or with different model data. As such, developers should use the listed environment variables in their scripts as much as possible. The coordinating team encourages developers to conduct the following robustness tests:

I. Test the POD with another set of model data

Do not even need to prepare another set of data. Developers can simply change the names of the model output files and directories to mimic a new dataset. The PODs should be able to handle this "new dataset" by simply changing *CASENAME* in *namelist*.

Developers can also try to change the variable names (e.g., change *PRECT* to *pr*). PODs should be fine with a modified *set_variable* script.

II. Change the directory containing POD scripts, etc.

For instance, if the *var_code* directory is moved, overwrite *VARCODE* should solve this. Similarly, if the *QBOi.EXP1.AMIP.001* sample data directory is in a different location, overwrite *DATADIR* (and *VARDATA* for *obs_data*). To change the directory for saving the diagnostic results produced by the code package? Overwrite *variab_dir*.

A remark on compound environment variables

For example, *DATADIR* (by default) is defined by 2 environment variables, i.e., *DATA_IN* and *CASENAME*, thus referred to as *compound*. The other example is *variab_dir*. If these variables are set to be overwritten in a namelist file simultaneously (e.g., as a remnant of previous runs), it may lead to a

[14] In theory, all environment variables can be overwritten. But the MDTF coordinating team recommends against overwriting *DIAG_HOME*, *DIAG_ROOT*, and *RGB* – the existing PODs may be broken unexpectedly.

conflict. The *mdtf.py* script will resolve this by overwriting only the compound environment variables while discarding the other two variables.

7. Summary of things to be aware of

Here is a short list of tips on implementation:

I. Structure of the code package – Implementing the constituent PODs in accordance with the structure described in sections 2 and 3 makes it easy to pass the package (or just part of it) to other groups.

II. Robustness to model file/variable names – Each POD should be robust to modest changes in the file/variable names of the model output; see section 5 regarding the model output filename structure, and section 6 regarding using the environment variables and robustness tests. Also, it would be easier to apply the code package to a broader range of model output.

III. Save intermediate output – Can be used, e.g. to save time when there is a substantial computation that can be re-used when re-running or re-plotting diagnostics. See section 3.I regarding where to save the output.

IV. Self-documenting – For maintenance and adaptation, to provide references on the scientific underpinnings, and for the code package to work out of the box without support. See step 5 in section 2.

V. Handle large model data – The spatial resolution and temporal frequency of climate model output have increased in recent years. As such, developers should take into account the size of model data compared with the available memory. For instance, the example POD *precip_diurnal_cycle* and *Wheeler_Kiladis* only analyze part of the available model output for a period specified by the environment variables *FIRSTYR* and *LASTYR*, and the *convective_transition_diag* module reads in data in segments.

VI. Basic vs. advanced diagnostics (within a POD) – Separate parts of diagnostics, e.g, those might need adjustment when model performance out of obs range.

VII. Avoid special characters (!@#$%^&*) in file/script name

Appendix 1: Environment Variables used by MDTF code package with their default settings

Variable name	Default setting	Where it is set (anything can be added to namelist to override settings in mdtf.py)	Description
CASENAME	QBOi.EXP1.AMIP.001	namelist (CASE entry 1)	Name of model run, default is downloadable sample model data
model	CESM	namelist (CASE entry 2)	Convention for variable names as determined by $VARCODE/util/set_variables_$model.py Currently options are CMIP, CESM, AM4.
FIRSTYR	1977	namelist (CASE entry 3)	First year of model data for this case (default for sample model data)
LASTYR	1981	namelist (CASE entry 4)	Last year of model data for this case (default for sample model data)
make_variable_tar	1	namelist	Flag to either make (1) or not make (0) a tar file of WKDIR upon package completion
verbose	1	namelist	Determines how much text output comes from the mdtf.py scripts: 0-minimal 1-normal 2-copious 3-debug level
test_mode	False	namelist	True = mdtf.py script reports what it would do instead of calling the actual packages False = mdtf.py calls all requested packages
NCARG_ROOT	none	namelist	Path to NCL installation
DIAG_HOME*	mdtf/MDTF_v2.0	mdtf.py	Where to invoke script % python mdtf.py namelist
DIAG_ROOT*	$DIAG_HOME/..	mdtf.py	Where mdtf distribution was installed, one directory above DIAG_HOME
DATA_IN	mdtf/inputdata	mdtf.py	Contains all data in sub-directories obs_data/, model/
DATADIR	$DATA_IN/model/$CASENAME	mdtf.py	Contains model data (in CASENAME directories)
VARDATA	$DATA_IN/obs_data	mdtf.py	Contains pre-digested observational data (in module directories)
WKDIR	$DIAG_HOME/wkdir	mdtf.py	Output from packages, created when run
VARCODE	$DIAG_HOME/var_code	mdtf.py	Source code, including util/ and all PODs
RGB*	$VARCODE/util/rgb	mdtf.py	Color tables used in NCL
variab_dir	$WKDIR/MDTF_$CASENAME	mdtf.py	Output directory

*Environment variables should not be changed.