

Global Model TestBed Single Column Model Technical Guide v1.0

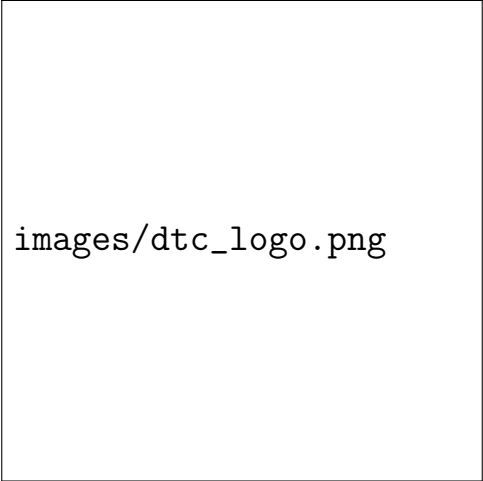
March 2018

Grant Firl, Laurie Carson

National Center for Atmospheric Research and Developmental Testbed Center

Ligia Bernardet, Dominikus Heinzeller

NOAA/ESRL Global Systems Division , Developmental Testbed Center and CIRES/CU



images/dtc_logo.png

Contents

Preface	iii
1 Introduction	1
1.1 Release Notes	1
1.1.1 Limitations	2
2 Quick Start Guide	3
2.1 Obtaining Code	3
2.2 System Requirements, Libraries, and Tools	3
2.2.1 Compilers	4
2.3 Compiling SCM with CCPP	4
2.4 Run the SCM with the supplied case	6
3 Repository	7
3.1 What is included in the repository?	7
4 Algorithm	9
4.1 Algorithm Overview	9
4.2 Reading input	9
4.3 Setting up vertical grid and interpolating input data	10
4.4 Physics suite initialization	10
4.5 Time integration	11
4.6 Writing output	11
5 Cases	12
5.1 How to run cases	12
5.1.1 Case configuration namelist parameters	12
5.1.2 Physics configuration namelist parameters	14
5.1.3 Case input data file	14
5.2 How to set up new cases	16
6 CCPP Interface	19
6.1 Setting up a suite	19
6.1.1 Preparing data from the SCM	19
6.1.2 Editing and running ccpp_prebuild.py	20
6.1.3 Preparing a suite definition file	20
6.2 Initializing/running a suite	21
6.3 Changing a suite	21

Preface

Meaning of typographic changes and symbols

Table 1 describes the type changes and symbols used in this book.

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories;	Edit your <code>.bashrc</code>
	on-screen computer output	Use <code>ls -a</code> to list all files.
AaBbCc123	What you type, contrasted with on-screen computer output	<code>host\$ You have mail!.</code> <code>host\$ su</code>
<i>AaBbCc123</i>	Command line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code>

Table 1: Typographic Conventions

1 Introduction

A single column model (SCM) can be a valuable tool for diagnosing the performance of a physics suite, from validating that schemes have been integrated into a suite correctly to deep dives into how physical processes are being represented by the approximating code. This SCM has the advantage of working with the Common Community Physics Package (CCPP), a library of physical parameterizations for atmospheric numerical models and the associated framework for connecting potentially any atmospheric model to physics suites constructed from its member parameterizations. In fact, this SCM serves as perhaps the simplest example for using the CCPP and its framework in an atmospheric model. The initial physics schemes included in the CCPP are the operational NOAA Global Forecast System (GFS) suite components that were implemented operationally in July 2017. The number of schemes that have met CCPP-compliance criteria is expected to grow significantly after the initial release. This expansion is expected to include many parameterizations to be considered for eventual operational implementation and their use within this model can provide evidence of performance improvement.

This document serves as the both the User and Technical Guides for this model. It contains a Quick Start Guide with instructions for obtaining the code, compiling, and running a sample test case, an explanation for what is included in the repository, a brief description of the operation of the model, a description of how cases are set up and run, and finally, an explanation for how the model interfaces with physics through the CCPP infrastructure.

Please refer to the release web page for further documentation and user notes.

<https://dtcenter.org/gmtb/users/ccpp/index.php>

1.1 Release Notes

The Bundle CCPP-SCM v1.0 contains the CCPP v1.0 and the GMTB SCM v2.0.

Physics parameterizations within v1.0 of CCPP are CCPP-compliant members of the operational 2017 GFS physics suite and include the following:

- GFS RRTMG shortwave and longwave radiation
- GFS ozone
- GFS Zhao-Carr microphysics
- GFS scale-aware mass-flux deep convection
- GFS scale-aware mass-flux shallow convection
- GFS hybrid eddy diffusivity-mass-flux PBL and free atmosphere turbulence

1 Introduction

- GFS orographic gravity wave drag
- GFS convective gravity wave drag
- GFS surface layer
- GFS Noah Land Surface Model
- GFS near-sea-surface temperature
- GFS sea ice
- Additional diagnostics and interstitial computations needed for the GFS suite

The CCPP framework contains the following

- Metadata standards for defining variables provided by the host application (in this case SCM) and needed by each parameterization
- `ccpp_prebuild.py` script to read and parse SCM and parameterizations metadata tables, compare the two and alert if incompatible, manufacture Fortran code for SCM and physics caps, and generate makefile snippets
- Suite Definition File that allows choosing parameterizations at runtime

GMTB SCM v2.0 is a major update to v1.3. The fundamental difference is how it calls physics. While v1.3 relied on IPDv2 from the GSM-based GFS to call physics, v2.0 calls physics through the CCPP framework. It includes the following:

- Cmake build system to compile needed NCEP libraries, SCM, CCPP framework, and parameterizations
- Physics variable metadata as part of a host model cap to the CCPP
- Test case of the Tropical Warm Pool - International Cloud Experiment (TWP-ICE)

1.1.1 Limitations

This release bundle has several known limitations:

- GMTB SCM v2.0 can only run one case. The other cases that had been working with v1.3 required specified surface fluxes, bypassing the surface schemes within the physics suite. This functionality will be restored in the next minor update, v2.1, which will include a new suite definition file that replaces the GFS surface schemes with a replacement scheme that “backs out” surface-related variables needed in PBL schemes from prescribed surface fluxes.
- The CCPP physics in this release only includes one scheme of each type, which makes it impossible to swap schemes within a suite. Additional schemes will be added soon to enable this functionality.
- The CCPP physics suite in this release contains several short “interstitial” schemes that mostly consist of short code sections that appeared in the antecedent GFS physics driver between calls to individual schemes. These were required to achieve bit-for-bit reproducibility between the physics called through the CCPP framework and those which were called through the IPDv4. The existence of these schemes limits the portability in the current release and these will be consolidated in a subsequent release to achieve greater portability of schemes.

2 Quick Start Guide

This chapter provides instructions for obtaining and compiling the GMTB SCM. The SCM code calls CCPP-compliant physics schemes through the CCPP framework code. As such, it requires the CCPP framework code and physics code, both of which are included as subdirectories within the SCM code. This package can be considered a simple example for an atmospheric model to interact with physics through the CCPP.

2.1 Obtaining Code

The source code bundle for the CCPP and SCM is provided using github.com. This release repository contains the tested and supported version for general use.

1. Download a compressed file or clone the source using

```
git clone https://[username]@github.com/NCAR/gmtb-scm-release
```

and enter your github password when prompted.

2. Change directory into the project.

```
cd gmtb-scm
```

The CCPP framework can be found in the ccpp-framework subdirectory at this level. The CCPP physics parameterizations can be found in the ccpp-physics subdirectory.

If you would like to contribute as a developer to this project, please see the Developers Guide at:

<https://dtcenter.org/gmtb/users/ccpp/developers/index.php>

2.2 System Requirements, Libraries, and Tools

The source code for the SCM and CCPP component is in the form of programs written in FORTRAN, FORTRAN 90, and C. In addition, the I/O relies on the netCDF libraries. Beyond the standard scripts, the build system relies on use of the Python scripting language, along with cmake, GNU make and date.

2 Quick Start Guide

The basic requirements for building and running the CCPP and SCM bundle are listed below:

- FORTRAN 90+ compiler
- C compiler
- cmake
- netCDF v3.6+
 - if netCDF v4.1+ is used, HDF5 and SZIP libs may also be required
- Python

Because these tools and libraries are typically the purview of system administrators to install and maintain, they are considered part of the basic system requirements.

There are several utility libraries provided in the SCM bundle, as external packages. These are built during the compilation phase, and include

- bacio
- sp
- w3
- w3nco

2.2.1 Compilers

The CCPP and SCM have been tested on a variety of computing platforms. Currently the CCPP system is actively supported on Linux and MacOS computing platforms using the Intel, PGI or GNU Fortran compilers. Unforeseen build issues may occur when using older compiler versions. Typically the best results come from using the most recent version of a compiler. The "Known Issues" section of the community website (https://dtcenter.org/gmtb/users/ccpp/support/CCPP_KnownIssues.php) provides notes regarding compiler support and known issues.

2.3 Compiling SCM with CCPP

The first step in compiling the CCPP and SCM is to match the physics variables, and build the physics caps needed to use them. Following this step, the top level build system will use cmake to query system parameters and make to compile the components. A platform-specific script is provided to load modules and set the user environment for common platforms. If you are not using one of these platforms, you will need to set up the same environment on your platform.

1. Run the CCPP prebuild script to match required physics variables with those available from the dycore (SCM) and to generate physics caps and makefile segments.
`./ccpp-framework/scripts/ccpp_prebuild.py`

- Note: add `-debug` to see the full output of the script.

2 Quick Start Guide

2. Change directory to the top-level SCM directory.

```
cd scm
```

3. (Optional) Run the machine setup script if necessary. This script loads compiler modules (Fortran 2003-compliant Intel), netCDF module, etc. and sets compiler environment variables.

For t/csh shell,

```
source etc/Theia_setup.csh or
source etc/Cheyenne_setup.csh or
source etc/UBUNTU_setup.csh or
source etc/CENTOS_setup.csh or
source etc/MACOSX_setup.csh
```

For bourne/bash shells,

```
. etc/Theia_setup.sh or
. etc/Cheyenne_setup.sh or
. etc/UBUNTU_setup.sh or
. etc/CENTOS_setup.sh or
. etc/MACOSX_setup.sh
```

- Note: If using a Mac, you may only need to run the MACOSX_setup script if you're following the instructions in doc/README_MACOSX.txt. Similar, if you are using Ubuntu/CentOS Linux, you may only need to run the UBUNTU_setup/CENTOS_setup script if you're following the instructions in doc/README_UBUNTU.txt / doc/README_CENTOS.txt. If your computing environment was previously set up to use modern compilers with an associated netCDF installation, it may not be necessary.

4. Make a build directory and change into it.

```
mkdir bin && cd bin
```

5. Invoke cmake on the source code to build.

```
cmake ../src (without threading/OpenMP)
cmake -DOPENMP=1 ../src (with threading/OpenMP)
```

6. Compile. Add VERBOSE=1 to obtain more information on the build process.

```
make
```

The resulting executable may be found at:

```
./gmtb-scm
```

If you encounter errors, please capture a log file from all of the steps, and contact the helpdesk at: gmtb-help@ucar.edu

2.4 Run the SCM with the supplied case

The test case provided with this version of the SCM is TWP-ICE, the Tropical Warm Pool-International Cloud Experiment. The SCM will go through the time steps, applying forcing and calling the physics defined in the suite definition file. There is a single command line argument required, which is the name of the case configuration file (without the .nml extension) found in `../etc/`. For the provided case, that name is `twpice`.

```
./gmtb_scm twpice
```

A netcdf output file is generated in the location specified in the case configuration file. Any standard NetCDF file viewing or analysis tools may be used to examine the output file (`ncdump`, `ncview`, `NCL`, etc). For the `twpice` case, it is located in:

```
gmtb-scm/scm/bin/output_twpice/output.nc
```

Additional details regarding the SCM may be found in the remainder of this Technical Guide and more information on the CCPP can be found in the CCPP Developers' Guide and at the following link: <https://dtcenter.org/gmtb/users/ccpp/index.php>.

3 Repository

3.1 What is included in the repository?

The repository contains all code and data required to run the GMTB SCM. It is functionally separated into 4 subdirectories representing the SCM model infrastructure (scm directory), the CCPP infrastructure (ccpp-framework directory), the CCPP physics schemes (ccpp-physics directory), and any necessary external library code (external directory). The entire gmtb-scm repository resides on github's NCAR account, and the ccpp-framework and ccpp-physics directories are git submodules that point to repositories on the same account. The structure of the entire repository is represented below.

```
gmtb-scm/
├── ccpp-framework/
│   ├── cmake/.....custom cmake code for building ccpp-framework
│   ├── CMakeLists.txt.....cmake configuration file for ccpp-framework
│   ├── doc/.....doxygen configuration, output, and User's Guide
│   ├── examples/.....contains suite definition files and XML schema definition files
│   ├── LICENSE
│   ├── README.md
│   ├── schemes/.....contains schemes used for testing and obsolete scripts
│   ├── scripts/. contains ccpp_prebuild and other python scripts for parsing metadata
│   └── src/.....contains CCPP framework code
├── ccpp-physics/.....contains all physics schemes and IPDv4-related code
│   ├── CCPP_CAPS.mk.....auto-generated makefile section for physics caps
│   ├── CCPP_schemes.mk.....auto-generated makefile section for physics schemes
│   ├── CMakeLists.txt.....cmake configuration file for ccpp-physics
│   ├── GFS_layer/.....driving routines used in IPDv4
│   ├── input.nml.....namelist containing parameters for operational GFS physics
│   ├── IPD_layer.....IPDv4 routines
│   ├── LICENSE
│   ├── makefile.....makefile to compile ccpp-physics used with FV3
│   ├── pgifix.py .3 physics/....contains all CCPP physics and interstitial schemes
│   │   └── docs/.....contains CCPP physics doxygen documentation
│   └── README.md
├── external/
│   ├── bacio.....NCEP library bacio (needed for GFS physics)
│   ├── sp.....NCEP library sp (needed for GFS physics)
│   └── w3nco.....NCEP library w3 (needed for GFS physics and SCM infrastructure)
└── scm/
    └── bin/.....build directory (initially empty; populated by cmake)
```

3 Repository

- └─ data/
 - └─ comparison_data/ contains data with which to compare SCM output
 - └─ GFS_physics_data/ contains data needed by the GFS physics suite
 - └─ processed_case_input/ ... contains initialization and forcing data for cases
 - └─ raw_case_input/ contains case data to be processed by scripts
 - └─ vert_coord_data/ contains data to calculate vertical coordinates
- └─ doc/ contains doxygen configuration file, images, and output
 - └─ html/ contains HTML output from doxygen
 - └─ images/ contains images used in the documentation
 - └─ README_MACOSX.txt
 - └─ README_UBUNTU.txt
 - └─ README_CENTOS.txt
- └─ etc/ contains case configuration, machine setup scripts, and plotting scripts
 - └─ case_config/ contains case configuration files
 - └─ Cheyenne_setup.csh setup script for Cheyenne HPC for csh, tcsh
 - └─ Cheyenne_setup.sh setup script for Cheyenne HPC for sh, bash
 - └─ gmtb_scm_ens.py example script for running an SCM forcing ensemble
 - └─ gmtb_scm_run.py example QSUB run script
 - └─ MACOSX_setup.csh setup script for Mac OS X for csh, tcsh
 - └─ MACOSX_setup.sh setup script for Mac OS X for sh, bash
 - └─ UBUNTU_setup.csh setup script for Ubuntu Linux for csh, tcsh
 - └─ UBUNTU_setup.sh setup script for Ubuntu Linux for sh, bash
 - └─ CENTOS_setup.csh setup script for CentOS Linux for csh, tcsh
 - └─ CENTOS_setup.sh setup script for CentOS Linux for sh, bash
 - └─ scripts/ python scripts for setting up cases and plotting
 - └─ f90nml.0.19/ f90nml python package
 - └─ plot_configs/ plot configuration files
 - └─ Theia_setup.csh setup script for Theia HPC for csh, tcsh
 - └─ Theia_setup.sh setup script for Theia HPC for sh, bash
- └─ LICENSE.txt
- └─ README.html link to doxygen generated documentation
- └─ src/ source code for SCM infrastructure
 - └─ cmake/ contains custom FindNetCDF.cmake

4 Algorithm

4.1 Algorithm Overview

Like most SCMs, the algorithm for the GMTB SCM is quite simple. In a nutshell, the SCM code performs the following:

- Read in an initial profile and the forcing data.
- Create a vertical grid and interpolate the initial profile and forcing data to it.
- Initialize the physics suite.
- Perform the time integration, applying forcing and calling the physics suite each time step.
- Output the state and physics data.

In this chapter, it will briefly be described how each of these tasks is performed.

4.2 Reading input

The following steps are performed at the beginning of program execution:

1. Call `get_config_nml()` in the `gmtb_scm_input` module to read in the `case_config` and `physics_config` namelists. This subroutine also sets some variables within the `scm_state` derived type from the data that was read.
2. Call `get_case_init()` in the `gmtb_scm_input` module to read in the case input data file. This subroutine also sets some variables within the `scm_input` derived type from the data that was read.
3. Call `get_reference_profile()` in the `gmtb_scm_input` module to read in the reference profile data. This subroutine also sets some variables within the `scm_reference` derived type from the data that was read. At this time, there is no “standard” format for the reference profile data file. There is a `select case` statement within the `get_reference_profile()` subroutine that reads in differently-formatted data. If adding a new reference profile, it will be required to add a section that reads its data in this subroutine.

4.3 Setting up vertical grid and interpolating input data

The GMTB SCM uses pressure for the vertical coordinate (lowest index is the surface). There are two choices for generating the vertical coordinate corresponding to the GSM-based GFS (set `model_name` = 'GFS' in the `case_config` file) and the FV3-based GFS (set `model_name` = 'FV3' in the `case_config` file). For both methods, the pressure levels are calculated using the surface pressure and coefficients (a_k and b_k). For the GSM-based vertical coordinate, the coefficient data is read from an external file. Only 28, 42, 60, 64, and 91 levels are supported. If using the FV3-based vertical coordinate, it is possible to use potentially any (integer) number of vertical levels. Depending on the vertical levels specified, however, the method of specification of the coefficients may change. Please see the subroutine `get_FV3_vgrid` in the source file `gmtb-scm/scm/src/gmtb_scm_vgrid.F90` for details. This subroutine was minimally adapted from the source file `fv_eta.F90` from the FV3 v0 public release version of the FV3 model.

After the vertical grid has been set up, the state variable profiles stored in the `scm_state` derived data type are interpolated from the input and reference profiles in the `set_state` subroutine of the `gmtb_scm_setup` module.

4.4 Physics suite initialization

For this model, it is possible to run several columns, each with a different physics suite. Therefore, suite initialization needs to be performed for each column. With the CCPP framework, initializing a physics suite is a 5-step process:

1. Call `ccpp_init()` with the path to the suite definition file and the CCPP derived data type (`cdata`) as arguments. This call will read and parse the suite definition file and initialize the `cdata` derived data type.
2. Initialize variables needed for the suite initialization routine. For suites based on the GFS operational suite, this involves setting some values in a derived data type used in the initialization subroutine and manually adding `ccpp_field_add()` calls for all data needed in the initialization subroutine.
3. Call `ccpp_run()` on the suite initialization subroutine, passing in the `cdata` derived data type (this subroutine should be listed in the `<init>` tag of the suite definition file).
4. Associate the `scm_state` variables with the appropriate pointers in the physics derived data type. Note: It is important that this step be performed before the next step to avoid segmentation faults.
5. Execute the `ccpp_field_add()` calls for the remaining variables to be used by the physics schemes. This step makes all physics variables that are exposed by the host application available to all physics schemes in the suite. This is done through an inclusion of an external file, `ccpp_fields.inc` that is automatically generated from the `ccpp_prebuild.py` script using the metadata contained in the host application cap (`/gmtb-scm/scm/src/gmtb_scm_type_defs.f90` in the current implementation).

4.5 Time integration

Two time-stepping schemes have been implemented within the GMTB SCM: forward Euler (`time_scheme = 1` in the `case_config` namelist) and filtered leapfrog (`time_scheme = 2` in the `case_config` namelist). If the leapfrog scheme is chosen, two time levels of state variables are saved and the first time step is implemented as a forward time step over $\frac{\Delta t}{2}$.

During each step of the time integration, the following sequence occurs:

1. Update the elapsed model time.
2. Calculate the current date and time given the initial date and time and the elapsed time.
3. If the leapfrog scheme is used, save the unfiltered model state from the previous time step.
4. Call the `interpolate_forcing()` subroutine in the `gmtb_scm_forcing` module to interpolate the forcing data in space and time.
5. Recalculate the pressure variables (pressure, exner, geopotential) in case the surface pressure has changed.
6. Call `do_time_step()` in the `gmtb_scm_time_integration` module. Within this subroutine:
 - Call the appropriate `apply_forcing_*` subroutine from the `gmtb_scm_forcing` module.
 - If using the leapfrog scheme, transfer the model state from one memory slot to the other.
 - For each column, call `ccpp_run()` to call all physics schemes within the suite (this assumes that all suite parts are called sequentially without intervening code execution)
7. If using the leapfrog scheme, call `filter()` in the `gmtb_scm_time_integration` module to time filter the model state.
8. Check to see if output should be written during the current time step and call `output_append()` in the `gmtb_scm_output` module if necessary.

4.6 Writing output

As of this release, the SCM output is only instantaneous. Specifying an `output_frequency` in the case configuration file greater than the timestep will result in data loss. Prior to the physics suite being initialized, the `output_init()` subroutine in the `gmtb_scm_output` module is called to create the netCDF output file and define all dimensions and variables. Immediately after the physics suite initialization and at the defined frequency within the time integration loop, the `output_append()` subroutine is called and instantaneous data values are appended to the netCDF file. Any variables defined in the `scm_state` and/or physics derived data types are accessible to the output subroutines. Writing new variables to the output involves hard-coding lines in the `output_init()` and `output_append()` subroutines.

5 Cases

5.1 How to run cases

Only two files are needed to set up and run a case with the SCM. The first is a configuration namelist file found in `gmtb-scm/scm/etc/case_config`. Each case configuration file contains two fortran namelists, one called `case_config` that contains parameters for the SCM infrastructure and one called `physics_config` that contains parameters for the physics suite(s). The second necessary file is a netCDF file containing data to initialize the column state and time-dependent data to force the column state. The two files are described below.

5.1.1 Case configuration namelist parameters

The `case_config` namelist expects the following parameters:

- `model_name`
 - This controls which vertical coordinates to use. Valid values are 'FV3' or 'GFS'. Here, 'GFS' refers to vertical coordinates used in the GSM.
- `n_columns`
 - The code can be used to run a single column or multiple *independent* columns using the same or different physics suites. Specify an integer, `n`.
- `case_name`
 - Identifier for which dataset (initialization and forcing) to load. This string must correspond to a dataset included in `gmtb-scm/scm/data/processed_case_input/` (without the file extension).
- `dt`
 - Time step in seconds (floating point)
- `time_scheme`
 - Specify 1 for the forward-Euler time-stepping scheme or 2 for the filtered leapfrog scheme.
- `runtime`
 - Specify the model runtime in seconds (integer). This should correspond with the forcing dataset used. If a runtime is specified that is longer than the supplied forcing, the forcing is held constant at the last specified values.
- `output_frequency`
 - Specify the frequency of the model output in seconds (floating point). Currently, no averaging of the output fields is done if `output_frequency` \neq `dt`; only instantaneous output at the supplied frequency is implemented.

5 Cases

- `n_levels`
 - Specify the integer number of vertical levels. If `model_name = 'GFS'`, only values of 28, 42, 60, 64, 91 are supported.
- `output_dir`
 - A string representing the path (relative to the build directory) to which output should be written.
- `output_file`
 - A string representing the name of the netCDF output file to be written (no `.nc` extension expected).
- `case_data_dir`
 - A string representing the path (relative to the build directory) where case initialization and forcing data files can be found.
- `vert_coord_data_dir`
 - A string representing the path (relative to the build directory) where vertical coordinate data files can be found (for `model_name = 'GFS'` only).
- `thermo_forcing_type`
 - An integer representing how forcing for temperature and moisture state variables is applied (1= total advective tendencies, 2= horizontal advective tendencies with prescribed vertical motion, 3= relaxation to observed profiles with vertical motion prescribed)
- `mom_forcing_type`
 - An integer representing how forcing for horizontal momentum state variables is applied (1= total advective tendencies, 2= horizontal advective tendencies with prescribed vertical motion, 3= relaxation to observed profiles with vertical motion prescribed)
- `relax_time`
 - A floating point number representing the timescale in seconds for the relaxation forcing (only used if `thermo_forcing_type = 3` or `mom_forcing_type = 3`)
- `sfc_flux_spec`
 - A boolean set to `.true.` if surface flux are specified from the forcing data (remove any surface schemes from the suite definition file if so)
- `sfc_type`
 - An integer representing the character of the surface (0: sea surface, 1: land surface, 2: sea-ice surface)
- `reference_profile_choice`
 - An integer representing the choice of reference profile to use above the supplied initialization and forcing data (1= "McClatchey" profile, 2: mid-latitude summer standard atmosphere)
- `year`
 - An integer representing the year of the initialization time
- `month`
 - An integer representing the month of the initialization time
- `day`
 - An integer representing the day of the initialization time
- `hour`
 - An integer representing the hour of the initialization time

5.1.2 Physics configuration namelist parameters

The `physics_config` namelist expects the following parameters:

- `physics_suite`
 - A string list representing the names of the physics suite for each column (must correspond a suite definition file name; if using multiple columns, you may specify an equal number of physics suites)
- `physics_suite_dir`
 - A string representing the path (relative to the build directory) where suite definition files can be found.
- `physics_nml`
 - A string list representing the paths (relative to the build directory) to the physics namelist files.
- `column_area`
 - A list of floating point values representing the characteristic horizontal domain area of each atmospheric column in square meters (this could be analogous to a 3D model's horizontal grid size or the characteristic horizontal scale of an observation array; these values are used in scale-aware schemes; if using multiple columns, you may specify an equal number of column areas)

5.1.3 Case input data file

The initialization and forcing data for each case is stored in a netCDF (version 4) file within the `gmtb-scm/scm/data/processed_case_input` directory. Each file has two dimensions (time and levels) and is organized into 3 groups: scalars, initial, and forcing. Not all fields are required for all cases. For example the fields `sh_flux_sfc` and `lh_flux_sfc` are only needed if the variable `sfc_flx_spec` = `.true.` in the case configuration file and state nudging variables are only required if `thermo_forcing_type` = 3 or `mom_forcing_type` = 3.

Listing 5.1: example netCDF file header for case initialization and forcing data

```
netcdf arm_sgp_summer_1997 {
dimensions:
    time = UNLIMITED ; // (233 currently)
    levels = UNLIMITED ; // (35 currently)
variables:
    float time(time) ;
        time:units = "s" ;
        time:description = "elapsed time since the beginning of the simulation" ;
    float levels(levels) ;
        levels:units = "Pa" ;
        levels:description = "pressure levels" ;

// global attributes:
        :description = "GMTB SCM forcing file for the ARM SGP Summer of 1997 case" ;

group: scalars {
} // group scalars

group: initial {
variables:
    float height(levels) ;
        height:units = "m" ;
```

5 Cases

```

        height:description = "physical height at pressure levels" ;
float thetail(levels) ;
    thetail:units = "K" ;
    thetail:description = "initial profile of ice-liquid water potential temperature" ;
float qt(levels) ;
    qt:units = "kg kg-1" ;
    qt:description = "initial profile of total water specific humidity" ;
float ql(levels) ;
    ql:units = "kg kg-1" ;
    ql:description = "initial profile of liquid water specific humidity" ;
float qi(levels) ;
    qi:units = "kg kg-1" ;
    qi:description = "initial profile of ice water specific humidity" ;
float u(levels) ;
    u:units = "m s-1" ;
    u:description = "initial profile of E-W horizontal wind" ;
float v(levels) ;
    v:units = "m s-1" ;
    v:description = "initial profile of N-S horizontal wind" ;
float tke(levels) ;
    tke:units = "m2 s-2" ;
    tke:description = "initial profile of turbulence kinetic energy" ;
float ozone(levels) ;
    ozone:units = "kg kg-1" ;
    ozone:description = "initial profile of ozone mass mixing ratio" ;
} // group initial

group: forcing {
    variables:
        float lat(time) ;
            lat:units = "degrees N" ;
            lat:description = "latitude of column" ;
        float lon(time) ;
            lon:units = "degrees E" ;
            lon:description = "longitude of column" ;
        float p_surf(time) ;
            p_surf:units = "Pa" ;
            p_surf:description = "surface pressure" ;
        float T_surf(time) ;
            T_surf:units = "K" ;
            T_surf:description = "surface absolute temperature" ;
        float sh_flux_sfc(time) ;
            sh_flux_sfc:units = "K m s-1" ;
            sh_flux_sfc:description = "surface sensible heat flux" ;
        float lh_flux_sfc(time) ;
            lh_flux_sfc:units = "kg kg-1 m s-1" ;
            lh_flux_sfc:description = "surface latent heat flux" ;
        float w_ls(levels, time) ;
            w_ls:units = "m s-1" ;
            w_ls:description = "large scale vertical velocity" ;
        float omega(levels, time) ;
            omega:units = "Pa s-1" ;
            omega:description = "large scale pressure vertical velocity" ;
        float u_g(levels, time) ;
            u_g:units = "m s-1" ;
            u_g:description = "large scale geostrophic E-W wind" ;
        float v_g(levels, time) ;
            v_g:units = "m s-1" ;
            v_g:description = "large scale geostrophic N-S wind" ;
        float u_nudge(levels, time) ;
            u_nudge:units = "m s-1" ;
            u_nudge:description = "E-W wind to nudge toward" ;
        float v_nudge(levels, time) ;
            v_nudge:units = "m s-1" ;
            v_nudge:description = "N-S wind to nudge toward" ;
        float T_nudge(levels, time) ;
            T_nudge:units = "K" ;
            T_nudge:description = "absolute temperature to nudge toward" ;
        float thil_nudge(levels, time) ;
            thil_nudge:units = "K" ;
            thil_nudge:description = "potential temperature to nudge toward" ;
        float qt_nudge(levels, time) ;
            qt_nudge:units = "kg kg-1" ;

```

5 Cases

```

        qt_nudge:description = "q_t to nudge toward" ;
float dT_dt_rad(levels, time) ;
        dT_dt_rad:units = "K s^-1" ;
        dT_dt_rad:description = "prescribed radiative heating rate" ;
float h_advec_thetail(levels, time) ;
        h_advec_thetail:units = "K s^-1" ;
        h_advec_thetail:description = "prescribed theta_il tendency due to horizontal advection"
float v_advec_thetail(levels, time) ;
        v_advec_thetail:units = "K s^-1" ;
        v_advec_thetail:description = "prescribed theta_il tendency due to vertical advection"
float h_advec_qt(levels, time) ;
        h_advec_qt:units = "kg kg^-1 s^-1" ;
        h_advec_qt:description = "prescribed q_t tendency due to horizontal advection" ;
float v_advec_qt(levels, time) ;
        v_advec_qt:units = "kg kg^-1 s^-1" ;
        v_advec_qt:description = "prescribed q_t tendency due to vertical advection" ;
} // group forcing
}

```

5.2 How to set up new cases

Setting up a new case involves preparing the two types of files listed above. For the case initialization and forcing data file, this typically involves writing a custom script or program to parse the data from its original format to the format that the SCM expects, listed above. An example of this type of script written in python is included in `/gmtb-scm/scm/etc/scripts/twipice_forcing_file_generator.py`. The script reads in the data as supplied from its source, converts any necessary variables, and writes a netCDF (version 4) file in the format described in subsection 5.1.3. For reference, the following formulas are used:

$$\theta_{il} = \theta - \frac{\theta}{T} \left(\frac{L_v}{c_p} q_l + \frac{L_s}{c_p} q_i \right) \quad (5.1)$$

$$q_t = q_v + q_l + q_i \quad (5.2)$$

where θ_{il} is the ice-liquid water potential temperature, θ is the potential temperature, L_v is the latent heat of vaporization, L_s is the latent heat of sublimation c_p is the specific heat capacity of air at constant pressure, T is absolute temperature in K, q_t is the total water specific humidity, q_v is the water vapor specific humidity, q_l is the suspended liquid water specific humidity, and q_i is the suspended ice water specific humidity.

As shown in the example netCDF header, the SCM expects that the vertical dimension is pressure levels (index 1 is the surface) and the time dimension is in seconds. The initial conditions expected are the height of the pressure levels in meters, and arrays representing vertical columns of θ_{il} in K, q_t , q_l , and q_i in kg kg^{-1} , u and v in m s^{-1} , turbulence kinetic energy in $\text{m}^2 \text{s}^{-2}$ and ozone mass mixing ratio in kg kg^{-1} .

For forcing data, the SCM expects a time series of the following variables: latitude and longitude in decimal degrees [in case the column(s) is moving in time (e.g., Lagrangian column)], the surface pressure (Pa) and surface temperature (K). If surface fluxes are specified for the new case, one must also include a time series of the kinematic surface sensible heat flux (K m s^{-1}) and kinematic surface latent heat flux ($\text{kg kg}^{-1} \text{m s}^{-1}$). The following variables are expected as 2-dimensional arrays (vertical levels first, time second): the geostrophic u (E-W) and v (N-S) winds (m s^{-1}), and the horizontal and vertical advective tendencies of θ_{il}

5 Cases

(K s⁻¹) and q_t (kg kg⁻¹ s⁻¹), the large scale vertical velocity (m s⁻¹), large scale pressure vertical velocity (Pa s⁻¹), the prescribed radiative heating rate (K s⁻¹), and profiles of u , v , T , θ_{il} and q_t to use for nudging.

Although it is expected that all variables are in the netCDF file, only those that are used with the chosen forcing method are required to be nonzero. For example, the following variables are required depending on the values of `mom_forcing_type` and `thermo_forcing_type` specified in the case configuration file:

- `mom_forcing_type = 1`
 - Not implemented yet
- `mom_forcing_type = 2`
 - geostrophic winds and large scale vertical velocity
- `mom_forcing_type = 2`
 - u and v nudging profiles
- `thermo_forcing_type = 1`
 - horizontal and vertical advective tendencies of θ_{il} and q_t and prescribed radiative heating (can be zero if radiation scheme is active)"
- `thermo_forcing_type = 2`
 - horizontal advective tendencies of θ_{il} and q_t , prescribed radiative heating (can be zero if radiation scheme is active), and the large scale vertical pressure velocity
- `thermo_forcing_type = 2`
 - θ_{il} and q_t nudging profiles and the large scale vertical pressure velocity

For the case configuration file, it is most efficient to copy an existing file in `gmtb-scm/scm/etc/case_config` and edit it to suit one's case. Recall from subsections 5.1.1 and 5.1.2 that this file is used to configure both the SCM framework parameters (including how forcing is applied) and some physics suite parameters. Be sure to check that model timing parameters such as the time step and output frequency are appropriate for the physics suite being used. There is likely some stability criterion that governs the maximum time step based on the chosen parameterizations and number of vertical levels (grid spacing). The `case_name` parameter should match the name of the case input data file that was configured for the case (without the file extension). The runtime parameter should be less than or equal to the length of the forcing data unless the desired behavior of the simulation is to proceed with the last specified forcing values after the length of the forcing data has been surpassed. The initial date and time should fall within the forcing period specified in the case input data file. If the case input data is specified to a lower altitude than the vertical domain, the remainder of the column will be filled in with values from a reference profile. There is a tropical profile and mid-latitude summer profile provided, although one may add more choices by adding a data file to `gmtb-scm/scm/data/processed_case_input` and adding a parser section to the subroutine `get_reference_profile` in `gmtb-scm/scm/src/gmtb_scm_input.f90`. Surface fluxes can either be specified in the case input data file or calculated using a surface scheme using surface properties. If surface fluxes are specified from data, set `sfc_flux_spec` to `.true..` Otherwise, specify a `sfc_type`.

To control the forcing method, one must choose how the momentum and scalar variable forcing are applied. The three methods of Randall and Cripe (1999, JGR) have been implemented: "revealed forcing" where total (horizontal + vertical) advective tendencies are

5 Cases

applied (type 1), “horizontal advective forcing” where horizontal advective tendencies are applied and vertical advective tendencies are calculated from a prescribed vertical velocity and the calculated (modeled) profiles (type 2), and “relaxation forcing” where nudging to observed profiles replaces horizontal advective forcing combined with vertical advective forcing from prescribed vertical velocity (type 3). If relaxation forcing is chosen, a `relaxation_time` that represents the timescale over which the profile would return to the nudging profiles must be specified.

The `physics_config` namelist portion of the case configuration file provides a place to specify what physics are called. One needs to specify the path where the CCPP suite definition file is located in the `physics_suite_dir` parameter. If one has specified that more than one (independent) columns be simulated, one also need to specify the name of the physics suite to be used for each column (corresponding to a suite definition file without the file extension) and a namelist that contains physics suite parameters for each column. The physics suite parameter namelist files are read as part of the suite initialization subroutine specified in the suite definition file. In addition, one must specify a `column_area` for each column. As of this release, multiple column functionality is limited to changes in the physics – one can run multiple different physics suites using the same initial conditions and forcing, perform sensitivity test through multiple physics suite parameter namelists, and/or sensitivity tests with different representative column areas.

6 CCPP Interface

Chapter 3 of the [CCPP Developers' Guide](#) provides a wealth of information on the overall process of connecting a host model to the CCPP framework for calling physics. This chapter describes the particular implementation within the GMTB SCM, including how to set up, initialize, call, and change a physics suite using the CCPP framework.

6.1 Setting up a suite

Setting up a physics suite for use in the GMTB SCM with the CCPP framework involves three steps: preparing data to be made available to physics through the CCPP, running the `ccpp_prebuild.py` script to reconcile SCM-provided variables with physics-required variables, and preparing a suite definition file.

6.1.1 Preparing data from the SCM

As described in sections 3.1 and 3.2 of the [CCPP Developers' Guide](#) a host model must allocate memory and provide metadata for variables that are passed into and out of the schemes within the physics suite. As of this release, in practice this means that a host model must do this for all variables needed by all physics schemes that are expected to be used with the host model. For the GMTB SCM, all variables needed by the physics schemes are allocated and documented in the file `gmtb-scm/scm/src/gmtb_scm_type_defs.f90` and are contained within the physics derived data type. This derived data type initializes its component variables in a `create` type-bound procedure. As mentioned in section 3.2 of the [CCPP Developers' Guide](#), a table containing all required metadata was constructed for describing all variables in the physics derived data type. The standard names of all variables in this table must match with a corresponding variable within one or more of the physics schemes. Appendix A of the [CCPP Developers' Guide](#) provides a list of all standard names used. The `local_name` for each variable corresponds to how a variable is referenced from the point in the code where `ccpp_field_add()` statements are made. For the GMTB SCM, then, all `local_names` begin with the physics derived data type. Nested within most of the `local_names` is also the name of a derived data type used within IPDv4 (re-used here for expediency). Since the `ccpp_field_add()` statements are made within a loop over all columns within `gmtb_scm.F90`, most `local_names` are also referenced with `i` as an array index.

6.1.2 Editing and running `ccpp_prebuild.py`

General instructions for configuring and running the `ccpp_prebuild.py` script can be found in section 3.4 of the **CCPP Developers' Guide**. The script expects to be run with a host-model-dependent configuration file. The `HOST_MODEL` variable within the script determines which configuration file is read. If `HOST_MODEL = "SCM"` (the default value in this release), the file `gmtb-scm/ccpp-framework/scripts/ccpp_prebuild_config_SCM.py` is used. Within this configuration file are variables that hold paths to the variable definition files (where metadata tables can be found on the host model side), the scheme files (a list of paths to all source files containing scheme entry points), the auto-generated physics schemes makefile snippet, the auto-generated physics scheme caps makefile snippet, the file where `ccpp_modules.inc` and `ccpp_fields.inc` are included, and the directory where the auto-generated physics caps should be written out to. Other variables less likely to be modified by a user are included in this configuration file as well, such as code sections to be included in the auto-generated scheme caps. As mentioned in section 2.3, this script must be run to reconcile data provided by the SCM with data required by the physics schemes before compilation by following step 1 in that section.

6.1.3 Preparing a suite definition file

The suite definition file is a text file read by the model at run time. It is used to specify the physical parameterization suite, and includes information about the number of parameterization groupings, which parameterizations that are part of each of the groups, the order in which the parameterizations should be run, and whether subcycling will be used to run any of the parameterizations with shorter timesteps.

In addition to the six or so major parameterization categories (such as radiation, boundary layer, deep convection, resolved moist physics, etc.), the suite definition file can also have an arbitrary number of additional interstitial schemes in between the parameterizations to prepare or postprocess data. In many models, this interstitial code is not known to the model user but with the suite definition file, both the physical parameterizations and the interstitial processing are listed explicitly.

The suite definition file also invokes an initialization step, which is run only once when the model is first initialized. Finally, the name of the suite is listed in the suite definition file. By default, this suite name is used to compose the name of the shared library (.so file) that contains the code for the physical parameterizations and that must be dynamically linked at run time.

For this release, the suite definition file used with the GMTB SCM is found in `gmtb-scm/ccpp-framework/examples/suite_scm_GFS_test.xml`. The suite has been named `"GFS_operational_2017"` and an initialization routine of `GFS_initialize_scm_run` has been specified. The physics schemes have been organized into 3 groupings following how the physics are called in the FV3GFS model, although no code is executed in the SCM time loop between execution of the grouped schemes. Several "interstitial" schemes are included in the suite definition file to execute code that previously was part of a hard-coded physics driver. Many of these schemes will eventually be rolled

into the schemes themselves, improving portability.

6.2 Initializing/running a suite

The process for initializing and running a suite in the GMTB SCM is described in sections 4.4 and 4.5, respectively. A more general description of the process for performing suite initialization and running can also be found in section 3.3 of the [CCPP Developers' Guide](#).

6.3 Changing a suite

When the CCPP has reached a state of maturity, the process for modifying the contents of an existing physics suite will be a very straightforward process, consisting of merely changing the name of the scheme in the suite definition file. As of this release, which consists of one scheme of each “type” in the pool of CCPP-compliant physics schemes with many short interstitial schemes, the process requires some consideration. A high-level guide for doing so is below.

- Examine and compare the arguments of the scheme being replaced and the replacement scheme (that is assumed to be CCPP-compliant).
 - Are there any new variables that the replacement scheme needs from the host application? If so, these new variables must be added to the host model cap. For the SCM, this involves adding a component variable to the physics derived data type and a corresponding entry in the metadata table. The new variables must also be allocated and initialized in the physics%create type-bound procedure.
 - Do any of the new variables need to be calculated in an interstitial scheme? If so, one must be written and made CCPP-compliant itself, and it must be added to the list of `scheme_names` in the `ccpp_prebuild.py` script configuration file. Further, it must be added to the suite definition file in the appropriate place.
 - Do other schemes in the suite rely on output variables from the scheme being replaced that are no longer being supplied by the replacement scheme? Do these output variables need to be derived/calculated in an interstitial scheme? If so, see the previous bullet about adding one.
- Examine existing interstitial schemes related to the scheme being replaced.
 - There may be scheme-specific interstitial schemes (needed for one specific scheme) and/or type-generic interstitial schemes (those that are called for all schemes of a given type, i.e. all PBL schemes). Does one need to write analogous scheme-specific interstitial schemes for the replacement?
 - Are the type-generic interstitial schemes relevant or do they need to be modified?
- Depending on the answers to the above considerations, edit the suite definition file as necessary (replacing the scheme itself and any needed interstitial scheme changes).
- If the replacement scheme is being added for the first time, edit the `ccpp_prebuild.py` configuration file by adding it and its associated interstitials to the `scheme_names` list.