Global Model Test Bed
Single Column Model (SCM)

User and Technical Guide
v2.1

August 2018
(updated March 2019)

Grant Firl, Laurie Carson

National Center for Atmospheric Research and Developmental Testbed Center

Ligia Bernardet, Dom Heinzeller

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

Acknowledgement

Contents

Contents

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; on-screen computer output	Edit your `.bashrc` Use `ls -a` to list all files. `host$ You have mail!.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`host$ su`
`AaBbCc123`	Command line placeholder: replace with a real name or value	To delete a file, type `rm filename`

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. The Global Model Test Bed (GMTB) 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 physics schemes included in the CCPP are the operational NOAA Global Forecast System (GFS) suite components that were implemented operationally in July 2017 and updates to the same suite as of August 2018, including the GFDL microphysics scheme. The number of schemes that have met CCPP-compliance criteria is expected to grow significantly in the near future. 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 v2.0 contains the CCPP v2.0 and the GMTB SCM v2.1.

Physics parameterizations within v2.0 of CCPP are CCPP-compliant members of the operational 2017 GFS physics and updates as of August 2018, including the GFDL microphysics scheme. The following schemes are included:

- GFS RRTMG shortwave and longwave radiation

- GFS ozone
- Choice of:
  - GFS Zhao-Carr microphysics
  - GFDL 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
- 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

GMTB SCM v2.1 is a minor update to v2.0. The fundamental differences are related to updates in the CCPP physics schemes since CCPP v1.0, the addition of new field campaign-based cases, and the capability to specify surface fluxes. 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
- the following test cases:
  - Tropical Warm Pool – International Cloud Experiment (TWP-ICE) maritime deep convection
  - Atmospheric Radiation Measurement (ARM) Southern Great Plains (SGP) Summer 1997 continental deep convection
  - Atlantic Stratocumulus Transition EXperiment (ASTEX) maritime stratocumulus-to-cumulus transition
  - Barbados Oceanographic and Meteorological EXperiment (BOMEX) maritime shallow convection
  - LES ARM Symbiotic Simulation and Observation (LASSO) for May 18, 2016 (with capability to run all LASSO dates - see 5.3) continental shallow convection

1.1.1 Limitations

This release bundle has some known limitations:

- The CCPP physics in this release includes one scheme of each type plus an additional microphysics scheme, so changing schemes is currently pretty limited. Additional schemes will be added soon to enable more of this type of functionality.
- The SCM cannot run over a land surface point using an LSM at this time due to the absence of necessary staged files that are used to initialize the LSM. Therefore, if the SCM is run over a land point (where sfc_type = 1 is set in the case configuration file), prescribed surface fluxes must be used:
  - surface sensible and latent heat fluxes must be provided in the case data file
  - sfc_flux_spec must be set to true in the case configuration file
  - the surface roughness length in cm must be set in the case configuration file
  - the suite defintion file used (physics_suite variable in the case configuration file) must have been modified to use prescribed surface fluxes rather than an LSM. An example is included in the development repository

  If the SCM is told to use an LSM (sfc_type = 1) without using prescribed surface fluxes as described above, the model will exit with a segmentation fault due to uninitialized variables. An update to use an LSM with the SCM will be forthcoming.
- The SCM is limited to running as a single column in this release. Multiple (independent) columns will be supported in the future.

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 through 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 gmtb-scm
   ```

   and enter your github password when prompted.
2. Check out the desired release number using

   ```
   git checkout [v2.0]
   ```

3. 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 CCPP Developers Corner of the project website:

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

There you will find links to all of the documentation pertinent to developers.

One can also clone the development repository, although its stability is not guaranteed. To do so,

1. Clone the source using

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

and enter your github password when prompted.

2. Check out the desired branch (master) using

   ```
   git checkout master
   ```

3. Change directory into the project.

   ```
   cd gmtb-scm
   ```

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.

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

- FORTRAN 90+ compiler (ifort v17+, gfortran v5.4+, pgf90 v17.9+)
- C compiler (icc v17+, gcc v5.4+, pgcc v17.9+)
- cmake v2.8.11+
- netCDF v4.x (not v3.x) with HDF5, ZLIB and SZIP
- Python v2.x (not v3.x)
- Libxml2 (tested with 2.2 and 2.9.1)

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 - Binary I/O Library
- sp - Spectral Transformation Library
- w3nco - GRIB decoder and encoder library

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. Please use versions listed in the previous section as unforeseen build issues may occur when using older compiler versions. Typically the best results come from using the most recent version of a compiler. If you have problems with compilers, please check the "Known Issues" section of the community website (https://dtcenter.org/gmtb/users/ccpp/support/CCPP_KnownIssues.php).

2.3 Compiling SCM with CCPP

The first step in compiling the CCPP and SCM is to match the physics variables (between what the host model – SCM – can provide and what is needed by physics schemes in the CCPP), 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 --config=./ccpp/config/
       ccpp_prebuild_config.py [--debug]
   ```

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), netCDF module, etc. and sets compiler environment variables. For *t/csh* shells,

   ```
   source etc/Theia_setup_gnu.csh
   source etc/Theia_setup_intel.csh
   source etc/Theia_setup_pgi.csh
   source etc/Cheyenne_setup_gnu.csh
   source etc/Cheyenne_setup_intel.csh
   source etc/Cheyenne_setup_pgi.csh
   source etc/UBUNTU_setup.csh
   source etc/CENTOS_setup.csh
   source etc/MACOSX_setup.csh
   ```

   For bourne/bash shells,

   ```
   . etc/Theia_setup_gnu.sh
   . etc/Theia_setup_intel.sh
   . etc/Theia_setup_pgi.sh
   . etc/Cheyenne_setup_gnu.sh
   . etc/Cheyenne_setup_intel.sh
   . etc/Cheyenne_setup_pgi.sh
   . etc/UBUNTU_setup.sh
   . etc/CENTOS_setup.sh
   . etc/MACOSX_setup.sh
   ```

   Note: If using a local Linux or Mac system, we provide instructions for how to set up your development system (compilers and libraries) in `doc/README_{MACOSX,UBUNTU,CENTOS}.txt`. If following these, you will need to run the respective setup script listed above. If your computing environment was previously set up to use modern compilers with an associated netCDF installation, it may not be necessary, although we recommend setting environment variables such as `CC`, `FC`, and `NETCDF`.

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=ON ../src              # with threading/OpenMP
cmake -DCMAKE_BUILD_TYPE=Debug ../src # debug mode
```

6. If cmake cannot find `libxml2` because it is installed in a non-standard location, add

```
-DPC_LIBXML_INCLUDEDIR=... -DPC_LIBXML_LIBDIR=...
```

to the cmake command.

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

```
make
```

The resulting executable may be found at `./gmtb-scm` (Full path of `gmtb-scm/scm/bin/gmtb-scm`). Depending on the system, it may be necessary to add the location of the CCPP framework and physics libraries to `LD_LIBRARY_PATH` to run `./gmtb-scm` (see next section).

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 a supplied case

There are several test cases provided with this version of the SCM. For all cases, the SCM will go through the time steps, applying forcing and calling the physics defined in the suite definition file. A single command line argument is required to run the model, which is the name of the case configuration file (without the .nml extension) found in `../etc/case_config/`. For a quick test that the model works, choose to run the `twpice` case:

```
./gmtb_scm twpice
```

If the run aborts with the error message

```
gmtb_scm: libccppphys.so.1.0.0: cannot open shared object file: No such
    file or directory
```

the environment variable `LD_LIBRARY_PATH` must be set to

```
export LD_LIBRARY_PATH=$PWD/ccpp/physics:$LD_LIBRARY_PATH
```

before running the model.

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 supplied `twpice.nml` case, it is located in:

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

or, simply,

`output_twpice/output.nc`

if you still reside in the `bin` directory from which you executed the model.

If using the model on HPC resources and significant amounts of processor time is anticipated for the experiments, it will likely be necessary to submit a job through the HPC's batch system. An example script has been included in the repository for running the model on Theia's batch system. It is located in `gmtb-scm/scm/etc/gmtb_scm_run.py`. Edit the `job_name`, `account`, etc. to suit your needs. The case name to be run is included in the `command` variable. To use, invoke

`./gmtb_scm_run.py`

from the `gmtb-scm/scm/etc` directory.

Additional details regarding the SCM may be found in the remainder of this guide. More information on the CCPP can be found in the CCPP Developers' Corner available at https://dtcenter.org/gmtb/users/ccpp/developers and in the CCPP Developers' Guide at https://dtcenter.org/gmtb/users/ccpp/docs.

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 `ccpp-framework` and `ccpp-physics` on the same account. Note, to facilitate the public release, the gmtb-scm-release repository also includes the `ccpp-framework` and `ccpp-physics` codes in `ccpp/framework` and `ccpp/physics`, so users do not have to obtain those as submodules. The structure of the entire repository is represented below. Note that the `ccpp_physics` repository contains files needed for using the CCPP with the experimental version of the Global Forecast System (GFS) that uses the Finite-Volume Cubed-Sphere (FV3) model (FV3GFS), even though the use of CCPP with FV3GFS is not supported in this release.

```
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 Developer's Guide
        ├── LICENSE
        ├── README.md
        ├── schemes/ ................................. contains schemes used for testing
        ├── scripts/ contains ccpp_prebuild and other python scripts for parsing metadata
        ├── src/ ...................................... contains CCPP framework code
        ├── test/ . contains scripts and configurations for testing .3 physics/ . contains all
          physics schemes
        ├── CCPP_CAPS.cmake .............. auto-generated cmake section for physics caps
        ├── CCPP_CAPS.mk ................ auto-generated makefile section for physics caps
        ├── CCPP_schemes.cmake ........ auto-generated cmake section for physics schemes
        ├── CCPP_schemes.mk .......... auto-generated makefile section for physics schemes
        ├── CMakeLists.txt ..................... cmake configuration file for ccpp-physics
        ├── LICENSE
        ├── pgifix.py
        ├── physics/ .................. contains all CCPP physics and interstitial schemes
```

```
        │   └─ docs/ ..................... contains CCPP physics doxygen documentation
        │   └─ stochastic_physics ... contains all stochastic physics code - FV3 only for now
        ├─ config/ .................................... contains the CCPP prebuild config
        ├─ suites/ ....................................... contains suite definition files
        │   └─ 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)
    │   ├─ 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
    │   │   ├─ README_MACOSX.txt
    │   │   ├─ README_UBUNTU.txt
    │   │   ├─ README_CENTOS.txt
    │   │   └─ TechGuide/ .......................... contains LaTeX for this User's Guide
    │   ├─ etc/ ........ contains case configuration, machine setup scripts, and plotting scripts
    │   │   ├─ case_config/ ........ contains case configuration files and physics namelist files
    │   │   │   └─ input_namelists/ ....... contains input namelists associated with the SDFs
    │   │   ├─ CENTOS_setup.csh ................ setup script for CentOS Linux for csh, tcsh
    │   │   ├─ CENTOS_setup.sh .................. setup script for CentOS Linux for sh, bash
    │   │   ├─ Cheyenne_setup_gnu.csh ......... setup script for Cheyenne HPC for csh, tcsh
    │   │   ├─ Cheyenne_setup_gnu.sh ........... setup script for Cheyenne HPC for sh, bash
    │   │   ├─ Cheyenne_setup_intel.csh ....... setup script for Cheyenne HPC for csh, tcsh
    │   │   ├─ Cheyenne_setup_intel.sh ........ setup script for Cheyenne HPC for sh, bash
    │   │   ├─ Cheyenne_setup_pgi.csh ......... setup script for Cheyenne HPC for csh, tcsh
    │   │   ├─ Cheyenne_setup_pgi.sh ........... setup script for Cheyenne HPC for sh, bash
    │   │   ├─ 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
    │   │   ├─ scripts/ ...................... python scripts for setting up cases and plotting
    │   │   │   ├─ f90nml.0.19/ .................................... f90nml python package
    │   │   │   └─ plot_configs/ .................................... plot configuration files
    │   │   ├─ Theia_setup_gnu.csh ................. setup script for Theia HPC for csh, tcsh
    │   │   ├─ Theia_setup_gnu.sh .................. setup script for Theia HPC for sh, bash
    │   │   ├─ Theia_setup_intel.csh .............. setup script for Theia HPC for csh, tcsh
    │   │   ├─ Theia_setup_intel.sh ............... setup script for Theia HPC for sh, bash
    │   │   ├─ Theia_setup_pgi.csh ................ setup script for Theia HPC for csh, tcsh
    │   │   ├─ Theia_setup_pgi.sh ................. setup script for Theia HPC 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
```

```
LICENSE.txt
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 a) the 2017 operational GFS based on the Global Spectral Model (GSM) (set `model_name` = 'GFS' in the `case_config` file), and b) the experimental 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 v0 release version of the FV3GFS 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

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 originating from the GFS model, this involves setting some values in a derived data type used in the initialization subroutine. Call the suite initialization subroutine to perform suite initialization tasks that are not already performed in the `init` routines of the CCPP-compliant schemes (or associated initialization stages for groups or suites listed in the suite definition file). Note: As of this release, this step will require another suite intialization subroutine to be coded for a non-GFS-based suite to handle any initialization that is not already performed within CCPP-compliant scheme initialization routines.
3. 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.
4. 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).

5. Call `ccpp_physics_init` with the `cdata` derived data type as input. This call executes the initialization stages of all schemes, groups, and suites that are defined in the suite definition file.

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 forward time step over $\Delta t/2$.

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 function, 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`. NOTE: As of this release, only `n_columns = 1` is supported.
- `case_name`
  - Identifier for which dataset (initialization and forcing) to load. This string must correspond to a dataset included in the directory `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.

- `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; not implemented yet, 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 (there is no need to have surface schemes in a 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" ;
    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 = "m^2 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" ;
```

19

```
      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" ;
     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/twpice_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:

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

$$q_t = q_v + q_l + q_i \tag{5.2}$$

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, 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 m^2 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 (kg kg^{-1} 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} (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.` and specify `sfc_roughness_length_cm` for the surface over which the column resides. 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 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 which 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.

5.3 Using other LASSO cases

In order to use other LASSO cases than the one provided, perform the following steps:

1. Access http://archive.arm.gov/lassobrowser and use the navigation on the left to choose the dates for which you would like to run a SCM simulation. Pay attention to the "Large Scale Forcing" tab where you can choose how the large scale forcing was generated, with options for ECMWF, MSDA, and VARANAL. All are potentially valid, and it is likely worth exploring the differences among forcing methods. Click on Submit to view a list of simulations for the selected criteria. Choose from the simulations (higher skill scores are preferred) and check the "Config Obs Model Tar" box to download the data. Once the desired simulations have been checked, order the data (you may need to create an ARM account to do

so).

2. Once the data is downloaded, decompress it. From the `config` directory, copy the files `input_ls_forcing.nc`, `input_sfc_forcing.nc`, and `wrfinput_d01.nc` into their own directory under **gmtb-scm/scm/data/raw_case_input/**.

3. Modify `gmtb-scm/scm/etc/scripts/lasso1_forcing_file_generator_gjf.py` to point to the input files listed above. Execute the script in order to generate a case input file for the SCM (to be put in **gmtb-scm/scm/data/processed_case_input/**):

   ```
   ./lasso1_forcing_file_generator_gjf.py
   ```

4. Create a new case configuration file (or copy and modify an existing one) in **gmtb-scm/scm/etc/case_config**. Be sure that the `case_name` variable points to the newly created/processed case input file from above.

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. A list of all standard names used can be found in `ccpp/framework/doc/DevelopersGuide/CCPP_VARIABLES_SCM.pdf`. 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 the FV3GFS cap (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, passed as argument `-config=path_to_config_file`. 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/suites/suite_SCM_GFS_2018_updated.xml`. 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

6.3.1 Replacing a scheme with another

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. Of course, prior to being able to swap a scheme within a suite, one must first add a CCPP-compliant scheme to the pool of available schemes in the CCPP physics repository. This process is described in section 2.2 of the CCPP Developers' Guide.

Once a CCPP-compliant scheme has been added to the CCPP physics repository, the process for modifying an existing suite should take the following steps into account:

- Examine and compare the arguments of the scheme being replaced and the replacement scheme.
  - 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. The CCPP Developers' Guide will help in this endeavor, and the process outlined in its section 2.2 should be followed.
  - 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. Typically, this would involve finding the `<scheme>` elements associated with the scheme to be replaced and its associated interstitial `<scheme>` elements and simply replacing the scheme names to reflect their replacements.

6.3.2 Modifying "groups" of parameterizations

The concept of grouping physics in the suite definition file (currently reflected in the `<group name="XYZ">` elements) enables "groups" of parameterizations to be called with other computation (perhaps related to the dycore, I/O, etc.) in between. In the suite definition file included in this release, three groups are specified, but currently no computation happens between `ccpp_run` calls for these groups. However, one can edit the groups to suit the needs of the host application. For example, if a subset of physics schemes needs to be more tightly connected with the dynamics and called more frequently, one could create a group consisting of that subset and place a `ccpp_run` call in the appropriate place in the host application. The remainder of the parameterizations groups could be called using `ccpp_run` calls in a different part of the host application code.

6.3.3 Subcycling parameterizations

The suite definition file allows subcycling of schemes, or calling a subset of schemes at a smaller time step than others. The `<subcycle loop = n>` element in the suite definition file controls this function. All schemes within such an element are called **n** times during one `ccpp_run` call. An example of this is found in the `suite_SCM_GFS_2018_updated.xml` suite definition file, where the surface schemes are executed twice for each timestep (implementing a predictor/corrector paradigm). Note that no time step information is included in the suite definition file. If subcycling is used for a set of parameterizations, the smaller time step must be an input argument for those schemes.

6.4 Adding variables

6.4.1 Adding a physics-only variable

Suppose that one wants to add the variable `foo` to a scheme that spans the depth of the column and that this variable is internal to physics, not part of the SCM state or subject to external forcing. Here is how one would do so:

1. First, add the new variable to the `physics` derived data type definition in `gmtb-scm/scm/src/gmtb_scm_type_defs.f90`. Within the definition, you'll notice

that there are nested derived data types (which contain most of the variables needed by the physics and are used for mainly legacy reasons) and several other integers/reals/logicals. One could add the new variable to one of the nested GFS derived data types if the variable neatly fits inside one of them, but it is suggested to bypass the GFS derived data types and add a variable directly to the `physics` type definition:

```fortran
real(kind=kind_phys), allocatable :: foo(:,:)
```

2. Second, within the `physics_create` subroutine, add an allocate and initialization statement.

```fortran
allocate(foo(n_columns, n_levels))
physics%foo = 0.0
```

Note that even though foo only needs to have the vertical dimension, it is also allocated with the `n_columns` dimension as the first dimension since this model is intended to be used with multiple independent columns. Also, the initialization in this creation subroutine can be overwritten by an initialization subroutine associated with a particular scheme.

3. At this point, these changes are enough to allocate the new variable (`physics%create` is called in the main subroutine of `gmtb_scm.F90`), although this variable cannot be used in a physics scheme yet. For that, you'll need to add an entry in the long metadata table entry that precedes the `physics` type definition in `gmtb_scm_type_defs.F90`. This entry looks like:

```fortran
!! | physics%foo(i) | foo | description of foo | units of foo |
   rank of foo | data type of foo| kind of data type (if real) |
   intent (none) | whether foo is optional (T or F) |
```

The elements of the metadata entry are all on one line (following the same format as other entries) and include the variable's "local name" or how it is referenced from `gmtb_scm.F90/main`, its "standard name" or how it is referenced by both the host model and the physics code, its units, its rank (dimensionality), Fortran intrinsic data type, the real kind if necessary, its intent (must be none for this host-side table), its optionality (must be F for this host-side table). This metadata entry is parsed by the CCPP framework and makes this variable available for any CCPP-compliant physics schemes to use.

4. On the physics scheme side, there will also be a metadata table entry for `foo` preceding the subroutine in which it is used. For example, say that scheme `bar` uses `foo`. If `foo` is further initialized in `bar`'s `_init` subroutine, a metadata entry for `foo` must precede the `bar_init` subroutine's code. If it is used in `bar`'s run subroutine, a metadata entry for foo must also appear in the preceding metadata table for `bar_run`. The metadata entry on the physics scheme side has the same format as the one on the host model side described above. The standard name, units, rank, type, and kind must match the entry from the host model table. Others attributes (local name, description, intent, optional) can differ. The local name corresponds to the name of the variable used within the scheme subroutine, and intent and optional attributes should reflect how the variable is actually used within the scheme.
Note: In addition to the metadata table, the argument list for the scheme subroutine must include the new variable (i.e., `foo` must actually be in the argument list for `bar_run` and be declared appropriately in regular Fortran).

If a variable is declared following these steps, it can be used in any CCPP-compliant

physics scheme and it will retain its value from timestep to timestep. A variable will ONLY be zeroed out (either every timestep or periodically) if it is in the `GFS_interstitial` or `GFS_diag` data types. So, if one needs the new variable to be 'prognostic', one would need to handle updating its value within the scheme, something like:

$$\text{foo}^{t+1} = \text{foo}^t + \Delta t * \text{foo\_tendency} \tag{6.1}$$

Technically, the host model can "see" foo between calls to physics (since the host model allocated its memory at initialization), but it will not be touching it.

6.4.2 Adding a prognostic SCM variable

The following instructions are valid for adding a passive, prognostic tracer to the SCM. Throughout these instructions, the new tracer is called 'smoke'.

1. Add a new tracer to the SCM state. In `gmtb-scm/scm/src/gmtb_scm_type_defs.f90` do the following:
   - Add an index for the new tracer in the `scm_state_type` definition.
   - Do the following in the `scm_state_create` subroutine:
     - Increment `scm_state%n_tracers`
     - Set `scm_state%smoke_index` = **(next available integer)**
     - Set `scm_state%tracer_names(scm_state%smoke_index)` = **'smoke'**
     - Note: `scm_state%state_tracer` is initialized to zero in this subroutine already, so there is no need to do so again.
2. Initialize the new tracer to something other than zero (from an input file).
   - Edit an existing input file (in `gmtb-scm/scm/data/processed_case_input`): add a field in the 'initial' group of the netCDF file(s) (with vertical dimension in pressure coordinates) with an appropriate name in one (or all) of the input netCDF files and populate with whatever values are necessary to initialize the new tracer.
   - Create a new input variable to read in the initialized values. In `gmtb-scm/scm/src/gmtb_scm_type_defs.f90`:
     - Add a new input variable in `scm_input_type`

       ```
       real(kind=dp), allocatable                :: input_smoke(:)
       ```

     - In `scm_input_create`, allocate and initialize the new variable to 0.
   - Read in the input values to initialize the new tracer. In `gmtb-scm/scm/src/gmtb_scm_input.f90/get_case_init`:
     - Add a variable under the initial profile section:

       ```
       real(kind=dp), allocatable  :: input_smoke(:) !< smoke
           profile (fraction)
       ```

     - Add the new input variable to the allocate statement.
     - Read the values in from the file:

       ```
       call check(NF90_INQ_VARID(grp_ncid,"smoke",varID))
       call check(NF90_GET_VAR(grp_ncid,varID,input_smoke))
       ```

- set `scm_input%input_smoke` = **input_smoke**
- Interpolate the input values to the model grid. Edit `gmtb_scm_setup.f90/set_state`:
  - Add a loop over the columns to call `interpolate_to_grid_centers` that puts **input_smoke** on grid levels in `scm_state%state_tracer`

    ```
    do i=1, scm_state%n_cols
        call interpolate_to_grid_centers(scm_input%
            input_nlev, scm_input%input_pres, scm_input%
            input_smoke, scm_state%pres_l(i,1,:), &
            scm_state%n_levels, scm_state%state_tracer(i
                ,1,:,scm_state%smoke_index,1),
                last_index_init, 1)
    end do
    ```

- At this point, you have a new tracer initialized to values specified in the input file on the model vertical grid, but it is not connected to any physics or changed by any forcing.

3. For these instructions, we'll assume that the tracer is not subject to any external forcing (e.g., horizontal advective forcing, sources, sinks). If it is, further work is required to:
   - One needs to provide data on how tracer is forced in the input file, similar to specifying its initial state, as above.
   - Create, allocate, and read in the new variable for forcing (similar to above).
   - Add to `interpolate_forcing` (similar to above, but interpolates the forcing to the model grid and model time).
   - Add statements to time loop to handle the first time step and different time-advancing schemes.
   - Edit `apply_forcing_forward_Euler` and `apply_forcing_leapfrog` in `gmtb-scm/scm/src/gmtb_scm_forcing.f90`.

4. In order to connect the new tracer to the CCPP physics, perform steps 1-4 in section 6.4.1 for adding a physics variable. In addition, do the following in order to associate the `scm_state` variable with variables used in the physics through a pointer:
   - Point the new physics variable to `scm_state%state_tracer(:,:,:,scm_state%smoke_index)` in `gmtb-scm/scm/src/gmtb_scm_type_defs.f90/physics_associate`.

5. There may be additional steps depending on how the tracer is used in the physics and how the physics scheme is integrated with the current GFS physics suite. For example, the GFS physics has two tracer arrays, one for holding tracer values before the physics timestep (`gmtb-scm/scm/src/GFS_typedefs.F90/GFS_statein_type/qgrs`) and one for holding tracer values that are updated during/after the physics (`gmtb-scm/scm/src/GFS_typedefs.F90/GFS_stateout_type/gq0`). If the tracer needs to be part of these arrays, there are a few additional steps to take.