[circa 2017-02-24; PRAMS_2.9]

Obtain and install/build the necessary code files

Prerequisites:
1) Fortran (95+) compiler (*e.g.*, gfortran 4.6+, ifort, pgf90)
2) C compiler (*e.g.*, gcc, icc, pgcc)
3) Python 2.6+ or Python 3.2+ (note that all Python scripts use '#!/usr/bin/env python')
4) NetCDF (must have been compiled with Fortran 90+ bindings; version 4+ is preferable; http://www.unidata.ucar.edu/software/netcdf/)

Optional Prerequisites:
5) MPI [*e.g.*, Open MPI (http://www.open-mpi.org) or MPICH2 (http://www.mcs.anl.gov/research/projects/mpich2/); must have been compiled with Fortran 90+ bindings] for parallel runs
6) NCL (http://www.ncl.ucar.edu/) for grid placement visualization
7) nco (http://nco.sourceforge.net/) for subsetting NetCDF-format GCM output (in time)

---

(1) Download and unpack the universal_lib source code tree

From a compressed archive: Decompress and extract the universal_lib archive in a directory of your choice (*e.g.*, */home/user/PRAMS* ; will automatically be unpacked into a subdirectory named *universal_lib)*:

```
bzip2 -dc universal_lib-1.2_r27-fs_dist.tar.bz2 | tar xvf - ;
```

-OR-

From the (private) bitbucket.org repository using git: Change to a directory of your choice (*e.g.*, */home/user/PRAMS*) and execute the below command (the placeholder *bitbucket_repository_URL* should be something like `https://some_user@bitbucket.org/some_user/universal_lib.git`). The repository contents will automatically be placed into a new subdirectory named *universal_lib*:

```
git clone {bitbucket_repository_URL};
```

---

(2) Configure and build the universal_lib source code tree

Change directory:

```
cd universal_lib/infrastructure/build/build_env_config;
```

Copy "*user_change_me-**" files most relevant to your computer system to this directory – for example:

```
cp examples/gfortran_gcc-linux/user_change_me-* . ;
```

Edit the "user_change_me-*" files as needed (*e.g.*, with specific compiler options), testing the success of the compilation via the following (iteration may be needed, along with inspection of the on-screen output and *../configure_build_env/work/config.log*):

```
../../../admin_script clean ALL; ../../../admin_script build debug;
```

NOTE: Only the NETCDF_* and MPI_* entries in *user_change_me-inclibs* need to be correct (for PRAMS, the NCL_NCARg_* and CFITSIO_* entries can be ignored).

If you encounter a compilation error involving something not found in module *mpi*, try adding "`-DBROKEN_MPI_MOD`" to your universal_lib *user_change_me-compilers.** files (then clean, and compile again).

---

(3) Download and unpack the PRAMS source code tree

From compressed archives: Decompress and extract the PRAMS code archives in a directory of your choice (*e.g.*, */home/user/PRAMS*; will automatically be unpacked into subdirectories named *common* and *Mars)*:

```
bzip2 -dc PRAMS_common-2.9_r29-fs_dist.tar.bz2 | tar xvf - ;
bzip2 -dc PRAMS_Mars-2.9_r29-fs_dist.tar.bz2 | tar xvf - ;
```

-OR-

From the (private) bitbucket.org repository using git: Change to a directory of your choice (*e.g.*, */home/user/PRAMS*) and execute the below (the placeholder *bitbucket_repository_URL_for_** should be something like `https://some_user@bitbucket.org/some_user/PRAMS.git`). The contents of the repository will automatically be placed into a new subdirectory named *PRAMS*:

```
git clone {bitbucket_repository_URL_for_PRAMS};
```

---

(4) Configure and build the PRAMS source code tree

Change directory:

```
cd Mars/infrastructure/build;
```

Make a copy of *build_env_config.other_packages-template* called *build_env_config.other_packages*, then optionally edit the new file appropriately (to specify where the relevant *universal_lib* and *common* directories are located):

```
cp build_env_config.other_packages-template
     build_env_config.other_packages;
```

Change directory:

```
cd build_env_config;
```

Copy the “user_change_me-*” files most relevant to your computer system to this directory – for example:

```
cp examples/gfortran_gcc-linux/* . ;
```

Edit the “*user_change_me-**” files as needed (*e.g.*, with specific compiler options), testing the success of the compilation via the following (iteration may be needed, along with inspection of the on-screen output and *common/infrastructure/build/configure_build_env/work/config.log*):

Change directory to *Mars*:

```
cd ../../..;
```

List the possible options available:

```
./admin_script -h;
```

Clean out any pre-existing build debris (in both universal_lib and PRAMS):

```
./admin_script clean ALL;
```

Typically, one would build the modeling system with one of the following commands:

```
./admin_script build;               ( serial)
./admin_script build DM_only;       ( parallel; MPI required)
./admin_script build debug;         (serial, with debugging flags)
./admin_script build DM_only debug; (parallel, with debugging flags; MPI required)
```

Install the desired static data files

This step does not necessarily have to be done every time – it is likely that one would not want too many copies of this on a single machine/filesystem, as these files (in total) are several GiB in size.

Decompress and extract the PRAMS data archives in a directory of your choice (*e.g.*, */data/user/input_static-PRAMS_2.9*; will automatically be unpacked into subdirectories named *common* and *Mars)*:

```
bzip2 -dc PRAMS_2.9-v1.common.full_data.tar.bz2 | tar xvf - ;
bzip2 -dc PRAMS_2.9-v3.Mars.smaller_data.tar.bz2 | tar xvf - ;
```

(**OPTIONAL**):

```
bzip2 -dc PRAMS_2.9-v1.Mars.large_data.tar.bz2 | tar xvf -;
```

Prepare the run directory

(1) This version of PRAMS offers a significant amount of flexibility regarding where its input and output data are located. However, in order to easily refer to those locations, it is suggested that a set of symbolic links pointing to those locations be created in the *run* directory. Also, in choosing a location for the PRAMS output, bear in mind that typical model output from a single PRAMS simulation can range in size from < 10 GiB to > 100 GiB, so ensure that the chosen directory resides on a data volume that can store significant quantities of data.

Change directory to *Mars/run*

Examples of creating such symbolic links:
`ln -s` *{dir_where_the_static_data_files_are}* `input_static;`
`ln -s` *{dir_where_the_GCM_output_data _are}* `MGCM_output;`
`ln -s` *{dir_for_PRAMS_output}* `output;`

(2) Copy relevant runscript template(s) from *example-runscripts* to *run*. For each of the three main types of PRAMS executables (prams, postp, prep), three choices of runscript are provided: (i) an executable runscript for use when running in serial (or on a machine without formal job queue management), (ii) a runscript with PBS directives (run_*.pbs; intended for use with ‘qsub’), and (iii) a runscript with sbatch directives (run_*.sbatch; intended for use with SLURM ‘sbatch’). For example:

On a machine without without formal job queue management (or for running in serial locally):
```
cp example-runscripts/run_PRAMS run_PRAMS;
cp example-runscripts/run_postp run_postp;
cp example-runscripts/run_prep run_prep;
```

On a machine with SLURM job queue management:
```
cp example-runscripts/run_PRAMS.sbatch run_PRAMS.sbatch;
cp example-runscripts/run_postp.sbatch run_postp.sbatch;
cp example-runscripts/run_prep.sbatch run_prep.sbatch;
```

(3) Copy relevant namelist template(s) from *example-namelists* to *run*. For example:

```
cp example-namelists/PRAMS_IN PRAMS_IN.test;
cp example-namelists/POSTP_IN POSTP_IN.test;
```

Updating the codebase(s)

With “official” compressed archive images: To update your codebase with an “official” archive image that you have obtained, use the install mode of the appropriate *admin_script* – note that the **.tar.bz2* can be in any directory, and will not be deleted or changed. <u>Be aware that any locally-modified source code with the same names will be overwritten</u>. Some examples:

```
cd PRAMS/PRAMS/common;
./admin_script install PRAMS_common-2.9_r30-fs_dist.tar.bz2;

cd PRAMS/PRAMS/Mars;
./admin_script install PRAMS_Mars-2.9_r30-fs_dist.tar.bz2;

cd PRAMS/universal_lib;
./admin_script install universal_lib-1.2_r28-fs_dist.tar.bz2;
```

-OR-

With the (private) bitbucket.org repository using git: A git-aware repository/directory (*i.e.*, `git status` doesn't return an error) already contains the bitbucket.org URL information, and can be updated as in the following examples (if you have any locally-modified source code, git may complain and suggest alternative courses of action – but that is beyond the scope of this guide):

```
cd PRAMS/PRAMS/common;
git pull;

cd PRAMS/PRAMS/Mars;
git pull;

cd PRAMS/universal_lib;
git pull;
```

Overview of typical use(s)

(1) Compile PRAMS (successfully) – see the above instructions/notes.

(2) **IF** using GCM initial state and boundary conditions [“INITIALIZATION_TYPE = 2” in the model namelist (*PRAMS_IN.something*)], preprocess the desired GCM output to be used. This process generates *proc_*.** files as output. For example:

```
./run_prep GCMOUTPUT*.nc;
```

To read about some available options:

```
./run_prep -h;
```

(3) Set up PRAMS grid placement/parameters in *PRAMS_IN.something*. To help visualize the grid placement, you may want to:

- Edit grid parameters in *PRAMS_IN.something*.
- Set "RUN_TYPE = 'GRID_INFO_FILE'" in *PRAMS_IN.something*.
- Run the model to create grid information files – it will exit after working briefly, and creates *grid_display-data.nc* in a directory based on the values of

"_OUTPUT_DIR_ROOT" and "SIMULATION_ID" in *PRAMS_IN.something* (*e.g.*, *./output/MPF_test-01/vis/*):

```
./run_PRAMS -f PRAMS_IN.something;
```

- Run an NCL visualization script, creating viewable charts in *PRAMS_grid_display.pdf* – for example:

```
./run_display_PRAMS_grids.py ./output/MPF_test-01/vis;
```

- View *PRAMS_grid_display.pdf* and decide if further changes to the grid are needed/desired. If so, repeat the steps in this sub-list.

(4) Set up all other PRAMS configuration parameters in *PRAMS_IN.something*.

(5) Create the PRAMS "surface files" (which contain the initial state of most ground surface characteristics). First, set "RUN_TYPE = 'GRID_INFO_FILE'" in *PRAMS_IN.something*. Then (this may take many minutes, depending on the grid/model configuration):

```
./run_PRAMS -f PRAMS_IN.something;
```

(6) **<u>IF</u>** using GCM initial state and boundary conditions ["INITIALIZATION_TYPE = 2" in the model namelist (*PRAMS_IN.something*)], create the PRAMS "var_files" (which contain the initial state of the model and time-dependent boundary conditions). First, set "RUN_TYPE = 'MAKE_VAR_FILES'" in *PRAMS_IN.something*. Then (this may take many minutes or even hours, depending on the grid/model configuration):

```
./run_PRAMS -f PRAMS_IN.something;
```

(7) Run the model, starting at the initial state and time. First, set "RUN_TYPE = 'INITIAL_START'" in *PRAMS_IN.something*. Then, for example:

(in serial):

```
./run_PRAMS -f PRAMS_IN.something;
```

-OR- to run (in serial) in the background without terminal interruption:

```
nohup ./run_PRAMS -f PRAMS_IN.something &> something.log &;
```

-OR- to run (in parallel), with 6 computational nodes and 1 supervisory node:

```
./run_PRAMS -n 7 -f PRAMS_IN.something;
```

-OR- to run (in parallel), using SLURM sbatch:
(i) first edit the top items (SBATCH directives, PRAMS command-line options, and text file to redirect stdout/stderr to) of *run_PRAMS.sbatch*
(ii) then

```
sbatch run_PRAMS.sbatch;
```

(8) Postprocess the model output (the raw PRAMS output is still in a somewhat arcane poorly-portable/readable format), in order to calculate certain derived variables, investigate a certain grid's results (or all of them), *et cetera*. First, edit *POSTP_IN.something* appropriately. Then run the postprocessor (only works in serial):

```
./run_postp -f POSTP_IN.something;
```

(9) Visualize/analyze the postprocessed model output in GrADS, ncview, Python, or whatever else you can make work.