GM RASP Install Procedure

Start with a functioning Linux. There are several Linux distributions to choose from. Then add the following packages:

Fedora

perl packages - install with yum:

```
sudo yum install perl-DateTime perl-Moose perl-Thread-Queue \
   perl-Set-Scalar \
   perl-File-Fetch perl-DateTime-Format-Strptime \
   perl-DateTime-Format-Duration perl-List-MoreUtils \
   perl-JSON perl-Net-Lite-FTP.noarch perl-CGI \
   perl-List-MoreUtils
```

Linux packages:

```
sudo yum install ImageMagick \
   ncl \
   gcc-gfortran \
   netcdf-fortran \
   mailx \
   jasper
```

Configure mailx

Error reports will be mailed to the admin user using mailx, so email must be enabled. If not already enabled, it may be sufficient to update the ~/.mailrc file:

```
set from=user@somedomain

set smtp=smtp_url
set smtp-auth-user=user@somedomain
set smtp-auth-password=password
```

Install GM RASP

Create an install directory:

```
mkdir rasp
```

Download RASP

```
cd rasp
git …
```

Get a site key from Google. They will want a credit card number, but unless your site is VERY popular, you'll never reach the point where they will actually charge you. Now, using your favorite editor, update 2 lines in HTML/index.html:

Update title in HTML/index.html as appropriate to describe your site:

```
<title>localhost RASP</title>
```

Update the site key using the key you got from Google:

```
&key=your_key
```

Install WRF/WPS executables in bin. They aren't provided in GM RASP. Note that GM RASP is tuned for a serial version of these tools on a dual processor system. A multi-processor version of these tools can actually take longer to complete a run. It's faster to run two serial WRF jobs simultaneously than to run two dual-processor jobs sequentially. Notes on how I build these executables are provided below.

Update site_params as appropriate for your site. The file should be self explanatory.

Add the following to ~/.bashrc for convenience

```
export RASPBASE=full_path-to-your-install-directory
export PATH=$PATH:$RASPBASE/bin
export LD_LIBRARY_PATH+=$RASPBASE/lib

alias torasp="cd $RASPBASE"
```

I added the following to crontab to automate running. Modify this as appropriate for your time zone and needs.

```
 0 21 * * * path_to_your_install_directory/bin/runit -t  1 -d
10 02 * * * path_to_your_install_directory/bin/runit -t  6 -d
 0 16 * * * path_to_your_install_directory/bin/runit -t 18 -d
```

At this point you should be able to run the provided sample regions:

```
bin/GM.pl -t 7 Panoch/nam
```

Note this runs a specific region/model/date. Running the entire suite takes several hours. Once this works, disable the sample regions in some way. One way is to rename the run.parameters files

```
for f in */*.run.parameters; do
mv $f ${f}+
done
```

Create New Region

Create a new top-level directory:

```
mkdir NewRegion
```

Copy prototype geogrid to new directory.

```
mkdir NewRegion/12x4km
cp NewEngland/12x4km/namelist.wps NewRegion/12x4km
```

namelist.wps is used to control the WPS utilities, including geogrid. This prototype uses a 12km grid for the outer domain, and a 4km grid for the inner domain. The inner domain is what will be plotted. Edit namelist.wps in the geogrid directory to represent the new geography. Start with ref_lat and ref_lon making them the center of the new region.

To view the new region,

```
cd NewRegion/12x4km
plotgrids
```

plotgrids will produce an image showing domain 1 with domain 2 as a white box inside. Examine the region enclosed by the white line - make it correct. Changing i_parent_start and j_parent_start will move the lower left corner of the white box. Changing e_we and e_sn will change the size of the boxes. See the online documentation for the geogrid parameters.

Alternatively the domain wizard can be used.

When everything is OK,

```
geogrid.exe
```

to create the .nc files. These are used during the run to build the input data for WRF.

To set up initial run parameters,

```
cp NewEngland/nam.run.parameters NewRegion
```

Edit as appropriate. Note that the plotTimes are Zulu, and day is number of days into the future to model, eg. changing day to 1 will cause tomorrow's forecast to be generated. Start testing/running

```
GM.pl NewRegion
```

Airspace files are located in the HTML sub-directory. The text version of the files have extension .kml, and in compressed form they are .kmz files. For use by Google Maps, they can be no larger than 3 MB - .kml or .kmz. When compressed, the uncompressed file can be no larger than 10 MB. The allusa file from the turnpoint exchange is MUCH larger than this. A script is provided to break this large file into smaller files that pass the size restriction.

The procedure is:

Download the All of US airspace file from the turnpoint exchange in GoogleEarth format. Put this file into the HTML sub-directory. Then, in the HTML sub-directory:

```
./split.sh
```

This will split the allusa file into 1 file per airspace type. Currently, five of these files are used. Should other airspace file(s) be desired, put them into the HTML sub-directory and edit the site_params file as appropriate. Processing these files involves Google reading them, so they need to be on a public web site. The airspaceBaseUrl variable in site_params can be used so that the files can be on a site different from the rasp server - useful when using localhost for site development.

Running GM RASP

GM.pl is the control code for building downloading the data files from NOAA, and processing them through WRF, plotting the results, and optionally pushing the results to the web server. The order of the arguments is unimportant.

```
GM.pl [jobarg*] [-d] [-t xx] [-v] [-s stage] [-h]
```

- jobarg is zero or more region/run.parameter references. If not specified, *.run.parameters from the current directory will be run. If that doesn't find any parameter files, $BINDIR/../*/*.run.parameters will be run, where $BINDIR is the location of the GM.pl script. Not providing any jobargs will normally run all the region/run.parameters jobs.
- If jobarg(s) are provided, they are passed through the bash filename expansion capabilities to find run.parameter files to run, ie. *jobarg*.run.parameters is used to find run.parameters files. If that fails, *jobarg*/*.run.parameters will be tried. Useful variations include:
  - NewEngland - will run all run.parameters files found in NewEngland
  - NewEngland/nam - will run NewEngland/nam.run.parameters
  - NewEngland/gfs* - will run all NewEngland/gfs run.parameters
- -d will purge results for days prior to today
- -t xx specifies the requested model run time (in zulu hours). The minutesand seconds in the current zulu time are zeroed, and the hour is set to the argument. Pre-pending the time with + or - signs will adjust the time by a day. For example, -t +5 says to set the run time to 5Z tomorrow. Specifying -t -5 says to use yesterday's data. Then for each weather model being run, the time is rounded down to the latest valid run time for that model. For example, specifying -t 7 for gfs or nam will result in using the results of the 06z run because these models only run every 6 hours. This same -t 7 will use the 07z run from rap or hrrr because they runevery hour.
- -v increases verbosity to the log output. -v -v increases it even more.
- -s stage says skip stages prior to the one specified, and start processing at the specified stage. This is useful for testing new features and control flow. For example, -s ungrib starts processing at the ungrib stage bypassing downloading data, assuming that data has already been downloaded. Note that this feature always requires that the corresponding downloads are available. Specifying -s wrf still requires the downloads to be available, even though they won't be used in the computations. Use the -h option to see the list of stage keywords.

Building executables

Notes on building WPS/WRF v4.0.3 on Fedora 28

```
sudo yum install jasper-devel libpng-devel \
     netcdf-fortran-devel libtirpc-devel
```

In performance testing, it was found that a shared-memory parallel build of wrf is only about 50% faster than a serial version. As a result, this version of RASP is tuned for using the serial build which results in actually completing a multi-day multi-model run quicker by running the tasks in parallel, ie. running 2 serial wrf's in parallel is faster than running them as parallel tasks, but sequentially one after the other.

Build WRF

```
mkdir netcdf
ln -s /usr/include netcdf/include
ln -s /usr/lib64 netcdf/lib
NETCDF=$PWD/netcdf WRFIO_NCD_LARGE_FILE_SUPPORT=1 ./configure
```

Choose gfortran, basic nesting
Apply Dr. Jack's patches.
Edit configure.wrf

- Add -ltirpc to the end of LIB_EXTERNAL
- Add the following line after `-I$(NETCDFPATH)/include \`

```
-I$(NETCDFPATH)/lib/gfortran/modules
```

Then build using:

```
CPATH="/usr/include/tirpc" ./compile em_real |& tee build.log
cp -p main/real.exe main/wrf.exe $RASPBASE/bin
```

Build WPS

```
mkdir netcdf
ln -s /usr/include netcdf/include
ln -s /usr/lib64 netcdf/lib
NETCDF=$PWD/netcdf ./configure
```

edit configure.wps to have -lnetcdff before the -lnetcdf

```
NETCDF=$PWD/netcdf ./compile |& tee build.log
cp -p geogrid/geogrid.exe ungrib/ungrib.exe \
     metgrid/metgrid.exe $RASPBASE/bin
```