
FRE: FMS Runtime Environment

Amy Langenhorst <Amy.Langenhorst@noaa.gov>

This is a howto document for the FRE (FMS Runtime Environment). The FRE is a tool to facilitate running FMS models. The user creates a model description file in xml format (or uses a preexisting file) and calls various FRE utilities which read it. The FRE utilities, written in perl, can acquire code, create and submit compile scripts, create and submit runscripts, and perform postprocessing, among other things. Balaji's document <http://cobweb.gfdl.noaa.gov/~vb/rts/describes> some goals, policies, and technical details of running regression tests with the FRE. A sample xml file which documents all the tags is available syntax-highlighted in HTML format at <http://www.gfdl.noaa.gov/fms/fre/example>. An "Introduction to FRE" powerpoint presentation is available at <http://cobweb.gfdl.noaa.gov/~arl/talks/frehowto/>. Fanrong Zeng has put together documentation on how to run CM2 experiments at <http://cobweb.gfdl.noaa.gov/~fjz/run-cm2/> and has an interface to viewing analysis figures for various experiments at <http://cobweb/fjz-cgi-bin/analysis/analysis.cgi>.

Note to external users: Paths in this document may be tailored to GFDL's site specifications. Also, the **frepp** utility is not available outside of GFDL.

Table of Contents

1. What is the FRE?	2
2. Quickstart Guide	2
3. Editing the XML	4
4. Inheritance	4
4.1. Special Case: Namelists	4
4.2. Special Case: Field Tables	5
4.3. Special Case: Executable	5
4.4. Special Case: mkmf template	5
5. Extending a run	5
6. Post-Processing	5
6.1. Check for missing postprocessing files offline	6
6.2. Creating postprocessing data or figures offline	6
6.3. Automated creation of diagnostic figures	6
7. Using totalview within the FRE environment	7
8. Using SMA within the FRE environment	8
9. Tips and Hints	8
10. Version History	9
A. Usage Information: fremake	9
B. Usage Information: frerun	9
C. Usage Information: frepp	10
D. Usage Information: frecheck	11
E. Usage Information: frelist	11
F. Usage Information: frestatus	11
G. Usage Information: frepriority	12
H. Usage Information: frescrub	12
I. Usage Information: freppcheck	13
J. fredb: coming soon	13
K. Naming conventions	13
1. CVS, Compilation Naming Conventions	13
2. Runscript Naming Conventions	13
3. Runscript Output Naming Conventions	14

1. What is the FRE?

The FRE consists of:

- An XML file which contains all experiment-specific variables such as the cvs commands used to check out the code, the path of the initial conditions file, namelists, and runtime specifications. The user needs to edit this file in order to customize the model configuration.
- C-shell template scripts for compiling and running models. The user does not need to look at them to run the FRE.
- Several conventions upon which defaults are based, such as the directory structure and naming conventions.
- Several perl scripts which read the XML file and perform a specific function:
 - **fremake**: checks out the model's code if necessary. Creates and optionally submits compile scripts using a c-shell compile template.
 - **frerun**: creates and optionally submits runscripts based on a c-shell runscript template.
 - **frepp**: creates and optionally submits postprocessing scripts. Can be launched automatically during a run or launched offline.
 - **frecheck**: runs reproducibility tests on output from FRE regression runs. Also calculates timing and performance statistics.
 - **frelist**: lists the experiments in your xml file, optionally with descriptive information about each model as provided in the xml.
 - **frestatus**: reports on the status of your batch compiles and batch runs. This information is parsed from the batch stdout files.
 - **frepriority**: provides job and queue control for long-running jobs.
 - **frescrub**: deletes duplicate copies of postprocessing output from your archive directory.
 - **freppcheck**: checks for missing postprocessing files
 - **fredb**: coming soon. It will provide an interface to the experiment database; for example, you will be able to add an experiment to the database with a call to fredb.

2. Quickstart Guide

This section describes how to obtain an xml file (current as of the last city release), compile an executable, perform short (regression test) runs, monitor the progress of the runs, and test the reproducibility of the output.

1. Acquire a FRE xml file. At GFDL, you can get the file containing experiments tested at the last city release by executing the following cvs checkout:

```
setenv CVSROOT /home/fms/cvs
cvs co rts
```

This will give you a directory called `rts/` and a file inside called `rts.xml`, which contains the xml for several experiments from the FMS Model Development Database.



Note

If you want to put your xml file in a location other than `$HOME/rts/rts.xml` and want your output in an archive location other than `$ARCHIVE/rts`, you will need to change the

following lines accordingly in your xml file. You can change the root directory, in which directories will be created for experiment code and scripts:

```
<directory type="root">$HOME/rts</directory>
```

and you can change the root location for your archive model output, in which subdirectories will be created here for each experiment's data:

```
<directory type="archive">$ARCHIVE/rts</directory>
```

2. Change to the directory containing `rts.xml` and run **frelist -v** to view available experiments. See [Appendix E, Usage Information: frelist](#) for usage information on **frelist**.



Note

You must either run the FRE utilities from the directory containing `rts.xml`, or else give them an `-x` argument with the path to your XML file.

You may need to add `/home/fms/bin`, or the directory containing the FRE utilities, to your Unix `$PATH`.

3. Run **fremake** only on the experiment(s) for which you want to check-out cvs code and/or compile. See [Appendix A, Usage Information: fremake](#) for usage information on **fremake**. For example, if you want to check-out and create a compile script for the experiments `am2p10` and `mom4_test1`, use: **fremake am2p10 mom4_test1**.



Note

The cvs source will be placed in `$root/$name/src/`. The compilation will be done in `$root/$name/exec/` and the executable will be created as `$root/$name/exec/fms_$name.x`.

Use the `-s` option to automatically qsub the compile script to the AC. The compile script's stdout will be placed in `$root/$name/exec/stdout`. You can monitor the progress of any FRE compile by monitoring the stdout file, or via **frestatus -c experiment**.

4. Run **frerun** on the experiment(s) you want to run. See [Appendix B, Usage Information: frerun](#) for usage information on **frerun**. To run all currently available regression tests (basic, scaling, restarts) on `am2p10` and `mom4_test1`, use: **frerun -r suite am2p10 mom4_test1**.



Note

Use the `-s` option to automatically qsub the scripts to the LSC.

The scripts' stdout files will be placed in `$archive/$name/$RUNPARAMS/ascii/stdout`, where `$archive` refers to the root archive location set in your xml file. In this example, `$archive` would be `$ARCHIVE/rts`. For details on the naming convention of `$RUNPARAMS`, see [Appendix K, Naming conventions](#).

5. Run **frestatus** to check the progress of the runs you've submitted as batch jobs. See [Appendix F, Usage Information: frestatus](#) for usage information on **frestatus**. To see how far your compiles or runs have gotten for `am2p10` and `mom4_test1`, use **frestatus am2p10 mom4_test1**.



Note

It is helpful to use the command **qa -n -u \$USER** along with **frestatus** to see which jobs are

running or waiting in the queue.

- When at least two regression test runs for a given experiment have completed (successfully), you can use **frecheck** to verify the reproducibility over pe-counts and restarts. See [Appendix D, Usage Information: frecheck](#) for usage information on **frecheck**. For example, to verify that the restart files for `am2p10` and `mom4_test1` have reproduced bit-for-bit across the scaling and restart regression tests, use: **frecheck am2p10 mom4_test1**.

3. Editing the XML

XML ([Extensible Markup Language](http://www.xml.com/pub/a/98/10/guide0.html) [<http://www.xml.com/pub/a/98/10/guide0.html>]) is a markup language similar to HTML, but where we've defined tags appropriate for our use in the FRE. The xml file is called `rts.xml` by default. This is a text file which you can edit with your favorite text editor. Vi (VIM), emacs, and nedit perform syntax highlighting automatically.

A sample xml file documents all the tags which are available. It is available syntax-highlighted in HTML format at <http://www.gfdl.noaa.gov/fms/fre/example>.

A tool called 'pollo' is available for browsing xml at `/home/arl/bin/pollo`. Pollo provides an interactive, graphical view of any xml file. Currently pollo can mess up your spacing if you save from it, so I recommend it only as an xml browser until a newer version is released. To use pollo to view a file, execute `/home/arl/bin/pollo rts.xml`.

4. Inheritance

It is possible for an experiment to inherit parameters from another experiment in the same xml file. Sample XML illustrating inheritance is shown in <http://www.gfdl.noaa.gov/fms/fre/example>. Aside from a few special cases, the rules governing inheritance are as follows.

- fremake** and **frerun** will look inside the experiment for the data, such as `<gridSpec>`, for example.
- If the data is not found, **fremake** and **frerun** will look for an `inherit` attribute in the `<experiment>` tag. If an `inherit` attribute is found, **fremake** and **frerun** will look inside the given experiment for the data. It will recurse in this manner until the data is found or until there are no more experiments from which to inherit.
- If the data is not found in the inheritance tree, the value will be empty. If the value was required, an error message will be printed. If the value was optional, a warning message will be printed if you use the `-v` option on **fremake** and **frerun**.

4.1. Special Case: Namelists

Namelists are parsed and will be given priority as follows:

- namelists directly in xml
- namelists in files, in the order the files are given in xml
- namelists inherited from parent experiment's xml
- namelists inherited from parent experiments' namelist files

A warning will be printed if you try to specify a namelist more than once in the inheritance tree, and an error will be printed if you try to give the same namelist twice directly in the xml tags for one experiment. It is currently not

possible to inherit only some values from a given namelist.

4.2. Special Case: Field Tables

Field tables are currently not parsed and are inherited on the basis of their file names. If you specify at least one field table, no field tables will be inherited from the parent experiment.

4.3. Special Case: Executable

The name of the executable has a default value if not specified anywhere. The default location is `$root/$name/exec/fms_$name.x`.

The FRE scripts decide whether a child experiment should have its own executable based on whether you have specified new data in the `<cvs>` or `<compile>` sections of the xml for your child experiment. If you intend for your experiment to inherit an executable, you should not re-specify anything in the `<cvs>` or `<compile>` sections of your child experiment, because then **fremake** will think you wanted to recompile with the new data. Actually, you do not need to run **fremake** on experiments which inherit an executable since the purpose of **fremake** is to create an executable.

4.4. Special Case: mkmf template

The location of the mkmf template is a special case because it has a default value if not specified anywhere. The default location is `/home/fms/bin/mkmf.template.$platform`.

5. Extending a run

After a run has finished, take the following steps:

1. Increase the length of the run in the xml to the full length of the run.
2. Use `frerun` to generate and submit a runscript.
3. Use `frepriority` to increase the number of queue allocations.

During a run,

1. Increase the length of the run in the xml to the full length of the run.
2. Use `frerun` to generate a runscript. `frerun` will overwrite the existing runscript. *Do not use the `-s` option or submit the script again.* The next time the script reloads, it will pick up the new version of the runscript.
3. Use `frepriority` to increase the number of queue allocations.

6. Post-Processing

One can create time series and climatological averages with the utility **frepp**. This can be called from the runscript as the model runs, or offline. Please note that in order to take advantage of the postprocessing, your runs should start on January 1, otherwise monthly and seasonal averages will not be able to be properly calculated. If you want to start your model run on another date, please run from that date to January 1, then start a new run including the postprocessing. Also, the postprocessing can handle segment lengths of 1 month, 6 months, or 1 year. It does not currently support other segment lengths.

For details on the xml syntax for requesting postprocessing, see <http://www.gfdl.noaa.gov/fms/fre/example>.

6.1. Check for missing postprocessing files offline

Fanrong Zeng has written a utility to check for missing postprocessing files. Call it as follows. It will ask you for the start year and end year you want to check.

```
freppcheck -x xmlfile experimentname
```

6.2. Creating postprocessing data or figures offline

It is fairly straightforward to submit the postprocessing offline if you know which arguments are needed. Here are some useful options; to see the full list, see [Appendix C, Usage Information: frepp](#).

```
-t year      = the (first) year of data to process
-p num      = "plus num years": additional years to process after the first y
-d dir      = path to history directory [default $archive/$name/history]
-A          = generate analysis figures only, based on existing pp data
```

To generate postprocessing data or figures based on someone else's experiment:

1. Copy their xml file to a directory you own. This will be the root location for postprocessing scripts.
2. Edit the <setup> section at the top of the xml file. The root directory should be the directory where you have placed the xml file, the archive directory should be the root location where you want postprocessing data to be placed, and the analysis directory should be the root location where you want analysis figures to be placed. Inside of the archive and analysis directories, a directory level with the experiment name will be created, and the pp directory will be created inside that.
3. Call **frepp**, making sure to use the **-d** option to specify the location of the original history files you want to process, ie, **-d /archive/user/cm2/cm2o_cmip/history**. For example, you might use a command like **frepp -d /archive/user/cm2/cm2o_cmip/history -t 0001 -p19 -x CM2.xml cm2o_cmip** to create the postprocessing data and figures for years 1-20 for experiment cm2o_cmip.

6.3. Automated creation of diagnostic figures

To automatically run your graphical scripts to create your own analysis with RTS, you need to convert your driver script to a template script by inserting the following lines at the beginning of your driver script.

```
#---- VARIBALES SET BY RTSP --##
set in_data_dir
set in_data_file
set descriptor
set out_dir
set WORKDIR

# plotting years
set yr1
set yr2
```

```
# data years, only used for making description file, only apply to ferret scripts using
# time series as input
set databegyr
set dataendyr
set datachunk

# Specify batch mode "batch" or interactive mode "interactive"
set mode

# Specify mom's version, either om2 or om3 because some files depend on mom's grid
set mom_version

# used as mask file
set gridspecfile

# used as mask file
set staticfile
```

Then put the template script in the analysis tag in a xml file like this:

```
<analysis switch="on" mode="batch" momGrid="om3"
  startYear="4-digit year number" endYear="4-digit year number"
  outdir="where you want to save your figure and text outputs"
  script="your template script with full path"/>
```

where

- **switch="on|off"** signals frepp to run|not run the analysis
- **mode="batch|interactive"** signals frepp to submit|not submit a script automatically
- **momGrid="om3|om2_173jrows|om2_174jrows"** specifies the mom grid if necessary,
- **startYear="XXXX"** specifies a specific year to start producing figures. The default value is the first year for which postprocessing data is available.
- **endYear="XXXX"** specifies a specific year to stop producing figures. The default value is the last year for which postprocessing data is available.
- **outdir=""** specifies the output directory where you want to save your figure and text output. You can also specify a directory with the **-O** command line argument to frepp, or with the **<directory type="analysis">** attribute in the setup section of your xml file. If none of the above methods are used, the figures will be placed in `/net2/user/analysis/$xpt`
- **script=""** is the template script that will be read by frepp.

Only the **script** attribute is required; all others are optional. The first value shown for the attribute is the default setting, followed by other valid options in descending priority. frepp reads your template script, specifies the variables, creates complete scripts for you and submits them if the mode is set to batch.

7. Using totalview within the FRE environment

Better support for totalview is coming soon; it's not very cleanly implemented yet. Currently, you need to

1. Use `/home/fms/bin/mkmf.debugtemplate.sgi` for compiling. I suggest a new experiment which inherits from the one you want to test, with

```
<compile>
  <mkmfTemplate>/home/fms/bin/mkmf.debugtemplate.sgi</mkmfTemplate>
</compile>
```

2. Generate a runscript with frerun and edit it... change

```
mpirun -np $npes $executable:t > fms.out
```

to

```
cd $executable:h
make localize
touch -t 197001010000 *.f90 *.F90 *.inc *.c *.h
cd $work
echo "totalview*searchPath: $executable:h" | xrdb -merge
totalview mpirun -a -np $npes $executable:t > fms.out
```

3. Run the script interactively.

8. Using SMA within the FRE environment

The `<cppDefs>` tag specifies `cppdefs` to use on the `mkmf` command line. The FRE also parses this section to determine whether you are using `libMPI` or `libSMA`. This information is then used to set up the runscript. The `<cppDefs>` section is required in your xml.

9. Tips and Hints

- You may wish to make other directories inside your `$root/$name` experiment directories, such as

```
$root
|-- am2p11
   |-- src    #created by fremake for cvs checkouts
   |-- exec  #created by fremake for compilation.
   |-- dev   #for code files under development. List in <srcList> in xml.
   |-- input #for input files like diag_table, etc.
```

- `qa -n 30` will give you a wider column for the experimentname.
- `<codeBase>` and `<modelConfig>` are used to construct the first cvs checkout command. The rest of the cvs commands go in `<cvsUpdates>`, which can handle any csh commands.
- You can use `$root` and `$name` in your xml elements. You will eventually also be able to set up other variables, but this has not yet been implemented.
- `frerun` sets the values of `$day` and `$month` in `coupler_nml`.
- om2 users: you can't just list the 'shared' directory currently in the `srcList` element because this list gets passed directly to `mkmf`, and `mkmf` doesn't search directories recursively. The optimal way to handle this is to check out `mom4`, rename the `path_names` file, and then check out the shared directory. The resulting `path_names` file will be used to compile shared code.
- If you want to run several configurations of the same model (using the same executable but different namelists or other input files) you don't need to check out and compile the same code multiple times. Compile once and use the 'inherit' attribute on the remaining experiments.

- For information on the naming of output directories, see [Appendix K, Naming conventions](#).

10. Version History

There is a version history for the FRE utilities at <http://www.gfdl.noaa.gov/fms/fre/version/>. Please let me know if you have problems or suggestions (Amy.Langenhorst@noaa.gov [mailto:Amy.Langenhorst@noaa.gov]).

Usage Information: fremake

Synopsis: fremake checks for the existence of the code directory for each experiment (root/experiment_name/src), and if it is not found, executes the cvs commands from your xml file. It then creates a simple compile script based on a c-shell template and variables from your xml file.

Usage: fremake [-Ffnsvthx:] experiment [experiment2 ...]

OPTIONS FOR COMPILING

-f = force fremake to run cvs commands, even if src directory exists
 -F = force delete of existing expt/src and expt/exec directories
 -n = don't run cvs commands, even if no src directory exists
 -t = use trap_uninitialized and other debugging FFLAGS
 -p = create path_names in single separate step after cvsUpdates; not d

GENERAL RTS OPTIONS

-h = show this help message, then exit
 -s = automatically submit the script with qsub
 -v = verbose flag
 -x xmlfile = use alternative xml file (default: rts.xml)
 experiment = experiment to create scripts for; must be found in xml file

Usage Information: frerun

Synopsis: frerun creates a runscript based on a runscript template and variables from an xml file. To perform regression tests, use the '-r' argument, otherwise frerun will look for a production tag for your experiment in your xml file.

Usage: frerun [-Obhnostuvx:r:] experiment [experiment2 ...]

OPTIONS FOR PRODUCTION RUNS

-o = overwrite any existing output directories
 -b = back up this production run to /archive/fms/fre_backup
 -O = back up only, creates a script which backs up a run

OPTIONS FOR REGRESSION TESTING

-r name = perform regression testing using the XML marked 'name'
 The keyword "suite" will do: basic, restarts, scaling.
 basic = one 8-day run
 scaling = one 8-day run on various pe-counts

```

        restarts = two 4-day runs and four 2-day runs on the
                    same number of pes as the 'basic' run
-t          = use the executable created with 'rtsmake -t' which uses
            trap_unititialized and other debugging FFLAGS
-o          = overwrite any existing output directories
-u          = create unique regression output directory names to save old data

RUNTIME UTILITIES
-n          = instead of creating a runscript, format and print a list of
            namelists in xml tags for the experiment

GENERAL RTS OPTIONS
-h          = show this help message, then exit
-s          = automatically submit the runscript(s) with qsub -l cpuset
-v          = verbose flag
-x xmlfile = use alternative xml file (default: rts.xml)
experiment = experiment to create scripts for; must be found in xml file

```

Usage Information: frepp

Synopsis: frepp01 is the RTS postprocessing utility.

Usage: frepp01 [-AO:c:d:ef:hp:oqr:st:u:vx:z] experiment [experiment2 ...]

OPTIONS FOR COMBINING MULTIPROCESSOR HISTORY FILES

-f filelist = list of files in \$archive/\$name/history to nc_combine

OPTIONS FOR POSTPROCESSING DATA

-t time = beginning of model year to process. To process 1982, use
 '-t 1982' or '-t 19820101'.

-p num = "plus num years": additional years to process after the first y

-c component = parallel pp; keyword 'split', or the component to be postproces

-d dir = path to history directory [default \$archive/\$name/history]

-u dir = specify unique output dir (other than 'pp')

OPTIONS FOR ANALYSIS FIGURES

-A = run analysis only

-O dir = where to put output figures. This argument is normally
 used with -A (run analysis only) and must be used if the
 xml file is not yours.

-Y year = specify a four digit year as your analysis's start year, ex -Y
 This year overrides the startYear specified in the analysis tag.

-Z year = specify a four digit year as your analysis's end year, ex -Z 19
 This year overrides the endYear specified in the analysis tag.

OPTIONS FOR TIMING/DEBUGGING/STATISTICS

-e = use old style history file timestamp at end time of file conten

-m = send mail to user when job is complete, ie qsub -m e

-q = don't print the error checks into the script. For evaluating c

-v = verbose flag

GENERAL RTS OPTIONS

-h = show this help message, then exit

-r runparams = regression test run. Combine netcdf files, but no further proc

-s = automatically submit the script with qsub

-x xmlfile = specify xml file (default: ./rts.xml)

experiment = experiment to create scripts for; must be found in xml file

Usage Information: frecheck

Synopsis: frecheck runs the resdiff command to compare restart files on the output produced by frerun-generated runs and prints a report. It also generates a table listing of the runtime for different processor counts by parsing the fms.out files.

Usage: frecheck [hlLnsvx:] experiment [experiment2 ...]

- c dir = output a script to compare output to experiments in this dir
- l = just list all the output cpio files. This is good for copying and pasting just the ones you want to test.
- L = just list all the output cpio files with dm status of files
- n = no dmgets
- s = save the tmp files created by resdiff
- h = show this help message, then exit
- x xmlfile = use alternative xml file (default: rts.xml)
- v = verbose
- experiment = experiment to check; must be found in xml file

Usage Information: frelist

Synopsis: frelist lists the experiments in your xml file.

Usage: frelist [e:hqvix:]

- e 'xpath' = xpath to information to retrieve for each experiment
example: -e 'input/gridSpec/@file'
note that this does not follow inherits.
- h = show this help message, then exit
- v = verbose (print experiment descriptions)
- q = quiet (do not print inheritance information)
- x xmlfile = use alternative xml file (default: rts.xml)

Usage Information: frestatus

Synopsis: frestatus parses batch script stdout to relay the status of your rtsmake and rtsrun shell scripts. Note that if you ran the scripts interactively, no status information will be found since there will be no batch script stdout files.

Usage: frestatus [-crhmpqtx:v] experiment [experiment2 ...]

```

-c          = show compile status only
-p          = show production statistics only
-r          = show run status only
-t          = show trapNAN executable information
-h          = show this help message, then exit
-x xmlfile = use alternative xml file (default: rts.xml)
experiment = experiment to get status of; must be found in xml file

```

Usage Information: frepriority

Synopsis: frepriority implements job control for RTS production runs. Without arguments, it will report job state, priority, and remaining queue allocations for the experiment.

```

Usage:    frepriority [ -a queue=num -g -s -P proj -h -x xmlfile ] experiment

-a queue=num = set [num] of job allocations for [queue], ie alloc=20
              [queue] must be either 'alloc' or 'windf'.
              CURRENTLY ONLY ONE -a argument per call to frepriority is parse
-g          = go; restart if job is in stop state
-s          = suspend; prevent the job from resubmitting itself
-P proj    = change priority/project, ie, '-P cmd'
-h          = show this help message, then exit
-v          = verbose option shows extra queue and reload information
-x xmlfile = use alternative xml file (default: ./rts.xml)
experiment = experiment to create scripts for; must be found in xml file

```

Usage Information: frescrub

Synopsis: frescrub writes a shell script to remove duplicate postprocessing data generated by FRE. With no capitalized arguments, it will print a report of postprocessing cpio files, netcdf files, and their sizes. You must choose one of the capitalized options to have frescrub write deletion commands into a shell script. The shell script will not be executed automatically unless you use the -s option. Also, frescrub will not delete any netcdf files if any errors were reported for their associated cpio file. To override this, use the -f option. You can specify year ranges between which to delete duplicate files with -i.

```

Usage:    frescrub [options] experiment [ experiment2 ... ]

CHOOSE ONE OR MORE WAYS TO SCRUB YOUR POSTPROCESSING
-A          = remove netcdf files when a cpio file exists
-B          = remove redundant, uncombined 'raw' history files
-C          = clean up $root by deleting excess old scripts
-D          = all of the above cleanup options

OTHER SCRUBBING OPTIONS
-f          = force scrub, overriding warnings
-d dir     = path to history directory (default $archive/$name/history)
-p dir     = path to pp directory (default $archive/$name/pp)

```

```

-i year-year = interval of dates to scrub: "-i 0001-0100" scrubs year 1-100
-c component = delete files for one component only

GENERAL RTS OPTIONS
-h           = show this help message, then exit
-s           = automatically submit the script with qsub
-v           = verbose option
-x xmlfile   = specify xml file (default: ./rts.xml)
experiment  = experiment to create scripts for; must be found in xml file

```

Usage Information: freppcheck

Synopsis: freppcheck checks for missing postprocessing files for a FRE experiment

Usage: freppcheck [options] experiment [experiment2 ...]

```

-T           = check all TS variables (by default, check only the first TS var
-c component = check only the specified component
-t time      = first year to check. If omitted, freppcheck will prompt you fo
-y time      = last year to check. If omitted, freppcheck will prompt you for
-h           = show this help message, then exit
-v           = verbose flag
-x xmlfile   = specify xml file (default: ./rts.xml)
experiment  = experiment to create scripts for; must be found in xml file

```

fredb: coming soon

Naming conventions

1. CVS, Compilation Naming Conventions

The program **fremake** creates a CVS checkout script in `$root/scripts/cvs_$name` where `$root` is defined at the top of your `rts.xml` file and `$name` is the experiment name. The CVS checkout script checks out source code into `$root/$name/src`.

The compilation script is then created and placed in `$root/scripts/mk_$name`. The executable created by the script will be `$root/$name/exec/fms_$name.x`.

2. Runscript Naming Conventions

The program **frerun** will create one or more runscripts. There are two methods of determining the name for a runscript based on whether you are running regression tests (**frerun** is invoked with the **-r regression_name** ar-

gument) or a production run (**frerun** is invoked without the **-r regression_name** argument).

For production runs, **frerun** will create the runscrip as `$root/scripts/$name`, deriving the runtime information from the `<production>` element(s) in your XML file. An example `<production>` element is shown here.

```
<runtime>
  <production simTime="8" units="years" npes="45">
    <segment simTime="1" units="months" runTime="00:44:00"/>
  </production>
</runtime>
```

The production runscrip will run the full simulation time of 8 years in 1 month segments as denoted above, re-starting itself as needed every 8 hours of run time. For explanation of how the production element XML is translated to runtime information in the runscrip, see <http://www.gfdl.noaa.gov/fms/fre/example>.

For regression tests, **frerun** will derive the runtime information from the `<regression>` element(s) in your XML file. Example `<regression>` elements are shown here.

```
<runtime>
  <regression name="basic">
    <run days="8" npes="15" runTimePerJob="00:30:00"/>
  </regression>
  <regression name="restarts">
    <run days="4 4" npes="15" runTimePerJob="00:20:00"/>
    <run days="2 2 2 2" npes="15" runTimePerJob="00:20:00"/>
  </regression>
  <regression name="scaling">
    <run days="8" npes="1" atmos_layout="1,0" ice_layout="1,0" runTimePerJob="04:00:00"/>
    <run days="8" npes="3" runTimePerJob="02:00:00"/>
    <run days="8" npes="45" runTimePerJob="00:20:00"/>
    <run days="8" npes="60" runTimePerJob="00:20:00"/>
  </regression>
</runtime>
```

To run a regression test with the information in the regression element labeled "basic" above, use **frerun -r basic \$name**. Then a runscrip will be created at `$root/scripts/$name_$runparams`, where `$runparams` is a string determined by the length of the run, the number of times the executable is called within the scrip, and the number of processors used. In the "basic" example above, `$runparams` would be `1x0m8d_15pe`, which translates to "one times zero months, eight days on 15 processors".

A single **frerun** command may create more than one runscrip for a given experiment. The runscrips will have different `$runparams` strings. With the example XML above, **frerun -r restarts \$name** would create two runscrips, and **frerun -r scaling \$name** would create four runscrips. The program **frerun** also recognizes the keyword `suite`, which would create runscrips from each of the three regression elements `basic`, `restarts` and `scaling`.

3. Runscrip Output Naming Conventions

Output directories are placed in `$archive/$name/` where `$archive` is specified at the top of your `rts.xml` file and `$name` is the experiment name. Production output is placed in three directories directly in `$archive/$name/`. For example, if you specified the following in your `rts.xml` file:

```
<setup>
  <directory type="archive">/archive/fms/fre</directory>
```

```
</setup>
```

then production output would be as follows for experiment am2p10:

```
/archive/fms/fre/am2p10/  
|-- ascii  
|-- history  
|-- restart
```

Regression test output utilizes another output directory level under `$archive/$name/` named for the runtime information as described above. The example regression elements shown in the previous section would produce the following output structure:

```
/archive/fms/fre/am2p10/  
|-- 1x0m8d_15pe  
|   |-- ascii  
|   |-- history  
|   |-- restart  
|-- 1x0m8d_1pe  
|   |-- ascii  
|   |-- history  
|   |-- restart  
|-- 1x0m8d_3pe  
|   |-- ascii  
|   |-- history  
|   |-- restart  
|-- 1x0m8d_45pe  
|   |-- ascii  
|   |-- history  
|   |-- restart  
|-- 1x0m8d_60pe  
|   |-- ascii  
|   |-- history  
|   |-- restart  
|-- 1x1m0d_45pe  
|   |-- ascii  
|   |-- history  
|   |-- restart  
|-- 2x0m4d_15pe  
|   |-- ascii  
|   |-- history  
|   |-- restart  
|-- 4x0m2d_15pe  
|   |-- ascii  
|   |-- history  
|   |-- restart
```