

# Rockin' in the FRE world

**Amy Langenhorst, Lori Thompson, Fanrong Zeng and V. Balaji**  
with input from Technical Systems Group (Bernie, Tim, ...)

**FMS Developers' Forum**  
**Princeton NJ**  
**24 May 2005**

# Capsule history of FRE

The FMS Runtime Environment began life as the FMS Regression Test Suite (RTS) in January 2003. Model development was in full swing, and experiments were being entered at a breakneck pace into the FMS Model Development database. The RTS offers a standard methodology for doing basic code integrity tests (restarts, PE counts) of new experiment configurations. The RTS is expressed in XML and included checkout, compilation, run and resubmit instructions (`rtsmake` and `rtsrun`).

As IPCC runs began, we introduced `rt spp`, a post-processing section that organized archive output in a manner reminiscent of the Climate group's former runscripts. It now included steps to interpolate atmospheric data onto standard pressure levels (Bruce Wyman's `plevel.sh`), time series and time average creation.

At this point, the project had gone beyond regression testing and had become a "CM2 production environment", and was renamed FRE.

# Elements of FRE

**fremake** Checkout an appropriate subset of the FMS source code for an experiment and create an executable;

**frerun** run an experiment in multiple *segments*; resubmit if necessary;

**frestatus** check the status of an experiment that is underway;

**frelist** list available experiments;

**frepriority** switch a job sequence between queues;

**frecheck** run RTS checks for bitwise accuracy;

**frepp** FRE post-processing: create time series, time averages, and plots;

**frescrub** remove intermediate and redundant files;

**freppcheck** RTS checks on history and post-processing files.

**freversion** tool to upgrade the XML, should the syntax change.

URL: <http://www.gfdl.noaa.gov/fms/fre>

# Why an overhaul?

- While it was always the intention to make FRE a general purpose environment, its initial deployment quickly became central to the lab's IPCC effort. Keeping the IPCC fires burning (or putting them out) became a full-time job. The time may now be right to consider a major rewrite, to make this usable in a wider range of settings.
- Issues with `/archive` over the last year led us to take a fresh look at the way FMS/FRE deals with large volumes of data;
- The earliest stage in model development – *prototyping* – requires a more nimble FRE than the RTS and production environment.
- Delivery of model data for collaboration has become a key activity. In IPCC, we encountered several “post-`frepp`” steps that could become part of the assembly line.
- Our alumni and their colleagues often request a “portable FRE”. A minimal “portable FRE” is part of our public release. There is also growing interest in the community in runtime environments and “standard” expressions of model configuration.
- FRE's many shortcomings.

# FRE's shortcomings

- The scripts FRE generates are too rigid: every runscript looks like a self-resubmitting, post-processing CM2 script.
- Too much intermediate data is archived.
- Reliance on inefficient, serial external utilities (NCO);
- Too much reliance on run segment lengths: a lot of the post-processing is triggered on month boundaries and so on.
- Every experiment generates a fresh checkout: shared codebase is not exploited where appropriate.
- Does not save enough state: there is not enough checking of when model configuration is altered.
- The XML, which most users edit by hand, is convoluted and verbose;
- The FRE perl scripts themselves could be more modular.

# Goals of the overhaul

**In general, each successive layer in software offers a reduced palette of choices with respect to the layer below.** This is good or bad depending on context. FRE-generated runs are probably most useful for development that is shared by a community at least as large and diverse as an MDT.

For such a community, FRE will offer:

- a *development*, testing and production environment for FMS-based models;
- that adequately serves the needs of CMDT, ESMDT, GAMDT, LMDT, OMDT and model liaisons;
- generates scripts that are no more (and no less) complex than they need to be;
- monitors performance and offers choices to reduce resource usage;
- and all the above being met, is abstract enough to be adapted by external users of FMS to their own needs.

Preliminary details of the design are being presented today. A more formal design document will be drafted based on today's feedback and subsequent discussions with interested parties. Implementation will take advantage of the rudimentary "public FRE" branch: we will prototype there and then merge into the current FRE trunk. Target delivery date is in late fall 2005, after the M-release.

## FRE's pattern of overlays

```
<namelist file="../../../input.nml"/>  
<namelist name="foo_nml">  
  .  
  .  
</namelist>
```

(1)

FRE often allows two ways of entering data: via a reference to an external file, and directly embedded in the XML. The direct entries overlie the imported elements.

This pattern is being extended to site configuration files. Many path names now hardcoded are now in an external `config.xml` file. Individual elements can be overridden in the `<setup>` section if required.

Everywhere where an external file is referenced, we will also permit a URL to be referenced. This is principally intended for direct integration of the model development database.

# Inheritance and state

This pattern allows us to generalize FRE's notion of *inheritance*. Currently, experiments inherit within the same file:

```
<experiment name="foo">  
  .  
  .  
</experiment>  
<experiment name="bar" inherit="foo">  
  .  
  .  
</experiment>
```

(2)

We will now permit inheritance across files.

**For all dependencies on external files, it is necessary to save one's state.** FRE currently performs all inheritance and overlays and saves the state of an XML node in an internal data structure (hash). We propose to write out each node hash as external well-formed XML as it is processed. The next time the same XML is processed, if there are any changes in state, the user will be alerted.

# FRE's model of storage resources

The current model of storage in FRE has two elements, **scratch** (`<directory type="work">`) and **archive** (`<directory type="archive">`). Archive storage on HPCS almost always points to `/archive`, a linear medium (tape). Linear media are markedly inefficient for non-linear use:

- random access patterns spanning non-contiguous regions of tape or multiple tapes;
- file deletion! (creates “holes” that are later filled in background tape defragmentation activity).

With a two-level storage resource model, it has been difficult to avoid non-linear use of linear media. We propose instead a three-level model:

**tmp** scratch space that is not guaranteed to exist beyond the end of a job;

**ptmp** random access storage that is not guaranteed to be backed up;

**archive** backed-up storage that is modeled as though it were “remote”, i.e. needing explicit fetch and store instructions.

During “prototyping” and “development” activities, there may be no need to archive much data. Even in production, much of what we deem to be **intermediate** data may not be archived. A **frearch** script (a mirror image of **frescrub**) will be provided to archive data in **ptmp**, should it become necessary. The use of **frescrub** on `/archive` will be needed less than now, and will be discouraged.

# Embedding custom script fragments

FRE currently allows a limited amount of embedded scripts:

```
<input>
  .
  .
  .
  <csH>
  .
  .
  .
  </csH>
</input>
```

(3)

We will extend and generalize this so that each XML node (<checkout>, <compile>, <input>, <run>, etc) can have a <csH> node inside anchored at the top or bottom of the section:

```
<input>
  .
  .
  .
  <csH anchor="top">
  .
  .
  .
  </csH>
</input>
```

(4)

Fragments can also be set to activate at, before or after a given value of model time:

```
<csH activateAfter="20050524103000">
  .
  .
  .
</csH>
```

(5)

This can be used to set shell variables that are later expanded within namelists, for instance.

## FRE's awareness of model time

One source of FRE frustration in early development has been its reliance on model *segments*, typically 1 or 6 months for AM2 and CM2 runs. As we expand on the use of triggers based on model time, as illustrated above, we will make this much more flexible. The syntax for this has not yet been developed.

FRE will not have too much knowledge of *calendars*: that is in the FMS *time\_manager*. It will rely on timestamps written by FMS.

## Extended suite of FRE programs

FRE now has dependencies on several external binaries, some homegrown and specialized (`plevel`, `timavg`), others more general (NCO utilities).

In general, we find that the NCO utilities take a kitchen-sink approach that is more designed for interactive use than batch processing of large data volumes. We seek to reduce our dependency on these by replacing with an equivalent set of specialized functions.

An important new function that will be added to the post-processing is the `xyInterp` attribute, to permit various subsampling and averaging options to reduce data volume.

```
<component type="ocean" xyInterp="subsample:2">  
<component type="ocean" xyInterp="average:2">  
<component type="ocean" xyInterp="regrid:/path/targetGridSpec.nc">
```

(6)

# Multiple compilation modes for a single experiment

FRE's interface to compile the same code in different ways is currently quite clunky.

- It has been a source of frustration for those running MOM scaling studies: MOM has a different compilation template for running in **DYNAMIC** mode, and **STATIC** at each PE count.
- FRE does not allow an easy switch to running a model under **totalview**.
- On Altix, regression testing requires an extra flag (**-fltconsistency**) that is turned off in production.

The new interface rationalizes these procedures.

```
<experiment>
  <executable>
    <compile exec="dynamic">
      <cppDefs>
    <compile exec="static60">
      <cppDefs>
    <run exec="dynamic">
  </experiment>
```

(7)