

---

# **ModelE-Control Documentation**

***Release 0.1***

**Elizabeth Fischer**

**Dec 22, 2018**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Design Requirements . . . . .	3
1.2	Advantages . . . . .	4
<b>2</b>	<b>Getting Started</b>	<b>7</b>
2.1	Quick Start: NCCS Discover . . . . .	7
2.2	Installation . . . . .	8
<b>3</b>	<b>Build and Run ModelE</b>	<b>11</b>
3.1	Setup the ModelE Root . . . . .	11
3.2	Download ModelE Source . . . . .	12
3.3	Setup ModelE Source . . . . .	12
3.4	Create a Run . . . . .	13
3.5	Start the Run . . . . .	14
3.6	View the Log . . . . .	15
3.7	Manage the Run . . . . .	16
3.8	Stop the Run . . . . .	16
3.9	Post-Mortem . . . . .	16
<b>4</b>	<b>Advanced Usage</b>	<b>19</b>
4.1	Elevation Classes . . . . .	19
4.2	Continuing a Run . . . . .	20
4.3	Restarting from fort.X.nc or an .rsf file . . . . .	20
4.4	Restarting by Date . . . . .	20
4.5	Keepalive . . . . .	21
4.6	Create a Rundeck . . . . .	21
4.7	One Root per User . . . . .	21
<b>5</b>	<b>Elevation Classes</b>	<b>23</b>
<b>6</b>	<b>Indices and tables</b>	<b>25</b>



ModelE Control makes it simple to setup, build and run the GISS ModelE climate model. It offers the following feature:

- ModelE run directories are decoupled from the ModelE software distribution, and may be placed anywhere convenient to the user. They will often reside together in “experiment” directories that might contains runs from more than one ModelE source download, as well as additional scripts to post-process experimental output.
- The ModelE software directory is not modified. This allows, for example, easy `rsync` of ModelE between computers.
- ModelE binaries are decoupled from source and build directories, and are only deleted when no longer used by any run. Re-building ModelE can never accidentally “mess up” an existing in-progress ModelE run.

Contents:



ModelE-Control is a new way to build and run ModelE. The original impetus of the system was to improve ModelE's ability to build with large numbers of external dependencies, in service to coupling with dynamic ice models. However, existing ways of running ModelE were not particularly easy or intuitive; the author would frequently forget how to do common tasks, or would make mistakes in submitting jobs.

## 1.1 Design Requirements

This experience led to a number of design goals:

1. There should be simple, high-level commands that do what users need with ModelE, without bothering them with the details. The list of required commands can be determined by looking at existing FAQs. For example:
  - How do I create a rundeck?
  - How do I prepare a run directory?
  - How do I run ModelE?
  - How do I stop ModelE?
  - How do I pause a ModelE run?
  - How do I restart a ModelE run somewhere other than it just left off?
2. ModelE runs last a long time. Changes a user might make to ModelE source code or its dependencies should not affect existing in-progress runs.
3. Users have diverse sets of needs on how ModelE is to be built and run; and they should all be supported. For example, some users run a single ModelE source code on dozens of rundecks. Some users run multiple versions of a ModelE on the same rundeck. Some users run specific rundecks on multiple compilers. Some users want to change a run in the middle, others run only versions with git hashes

All these use cases should be supported equally, without making specific assumptions on how users wish to work.

4. ModelE runs should be reproducible. Enough information should be collected in a run so it can be reproduced.

5. A stacktrace should be provided whenever a ModelE run terminates prematurely, for whatever reason. This can save countless hours of debugging effort.
6. Users update rundecks, and upstream changes to ModelE also update rundecks. These multiple sources of changes should be accommodated and merged gracefully, without forcing the user to manually merge them (except in case of conflict).
7. Modern software ecosystems demand an ever-increasing number of dependencies be installed for a project to build and run; avoiding dependencies is no longer a viable way to make software easy to install. It should be easy to install ModelE dependencies, and developers should not be inhibited from using valuable third-party libraries simply because they add a dependency.
8. Software dependencies of ModelE change from time to time — for example, NetCDF. Upgrading these dependencies should be easy and non-disruptive, and should not affect existing in-progress ModelE runs.
9. ModelE binaries should be useful for more than just running a climate simulations. For example, unit tests, single-system runs or single-column models should be possible, without having to change the core ModelE `main()` program.

ModelE-Control addresses these design goals by re-thinking the process of building and running ModelE. It is an evolution of existing practice and scripts. The result is a single `ectl` command, like `git`, with sub-commands for all ModelE operations.

## 1.2 Advantages

Before we dive into using ModelE Control, this section explains a few of the main concepts and advantages offered by the system.

### 1.2.1 Directories

In running ModelE, ModelE-Control distinguishes between five different directories:

- **root**: An umbrella directory containing multiple run directories.
- **run**: The ModelE run directory; that is, the directory in which ModelE output and other data files are written.
- **src**: The location of the ModelE source code, as downloaded from Git. ModelE-Control does not modify the `src` directory.
- **build**: The location where ModelE source code is built, which always happens out-of-source with ModelE-Control.
- **package**: The location for ModelE binaries.

The `run` directory is central to `modele-control`, commands all operate on a run directory. Within the run directory are symbolic links to the source (`src`), build (`build`) and package (`pkg`) directories currently associated with that run. This use of symbolic links to associate directories with each other has many advantages:

1. Users have flexibility to place related runs together, whether or not they were built from the same source.
2. Different runs using the same source but different rundecks will build in different build directories, limiting the need for large rebuilds.
3. Packages are guaranteed to last at least as long as the run that needs them, no matter what the user does to the associated source or build directory in the meantime.



### 1.2.2 Rundeck Management

Users typically start with a rundeck supplied in the ModelE repository, and then modify it as needed. This works, until ModelE is updated, and the rundeck templates with it. At that point, the user is left with a new rundeck that works but without modifications; and an old rundeck that no longer works. The user is forced to manually re-apply the edits made to the old rundeck, to the new rundeck.

ModelE Control mostly eliminates the need to manually merge rundecks. When a source directory is updated, ModelE Control will use Git to apply the user's rundeck modifications to the new rundeck. In case a rundeck changes in the middle of a run, this also allows the user to reconstruct when that change happened.

#### **/ File Management**

Sometimes, users need to change rundeck parameters in the middle of a run. In the past, that was done by modifying the *I* file. This was not user friendly because the *I* file is not the same as the original rundeck. With ModelE Control, the user can edit the rundeck directly when making parameter changes.



## CHAPTER 2

---

### Getting Started

---

A number of pieces of software must be installed to run ModelE successfully — not just ModelE and ModelE-Control, but also ModelE dependencies, as well as key post-processing tools. These tools have already been installed on some systems, allowing users to get started immediately simply by making changes to *.bashrc*.

Check below to see if the ModelE environment has been installed on your favorite computer. If not, head to the full Installation instructions below.

### 2.1 Quick Start: NCCS Discover

The ModelE environment has already been installed on NCCS Discover, allowing users to get started quickly with ModelE. To use this environment:

1. Remove all `module load` commands from your *.bashrc* file.
2. Add the following to your *.bashrc* file:

```
.. code-block:: console
```

```
source /home/rpfische/env/modele-ksx-gcc
```

Use the following instead for Intel compilers (not yet implemented):

```
.. code-block:: console
```

```
source /home/rpfische/env/modele-ksx-intel
```

3. Set `MODELE_FILE_PATH` in your *.bashrc*, modifying depending on where you wish to keep user-generated input files. For example:

```
export MODELE_FILE_PATH=/discover/nobackup/projects/giss/prod_input_files:$HOME/  
↪modele_input/local
```

That's it, you are now ready to use ModelE, along with all the latest tools, NetCDF, etc.

## 2.2 Installation

ModelE and ModelE-Control have a number of dependencies, which are automatically handled by [Spack](#). The following instructions may be used to install ModelE, `modele-control` and Spack starting from a clean machine:

### 2.2.1 Install Spack

If you are not using `discover`, then here is how to build a ModelE environment on your machine.

1. Download:

```
cd ~
# git clone git@github.com:citibeth/spack.git -b efischer/develop
git clone https://github.com/citibeth/spack.git -b efischer/develop
```

2. Add to your `.bashrc` file:

```
export SPACK_ROOT=$HOME/spack
. $SPACK_ROOT/share/spack/setup-env.sh
```

3. Remove non-system stuff from your `PATH`, `LD_LIBRARY_PATH` and other environment variables, which can cause strange errors when building with Spack.

### 2.2.2 Setup Spack

See Spack docs for more info on setting up compilers, bootstrapping, etc.

Actuallyxx... give instructions here on Intel, etc. compilers

### 2.2.3 Setup Packages

Copy the following to your `~/ .spack/packages.yaml` file:

```
packages:
  python:
    # Lie about the version in the system, so it's new enough for Spack to
    ↪recognize.
    paths:
      python@2.7.8: /          # Spack can't install Python2, I don't know why
    version: [3.5.2,2.7.8]

  py-cython:
    version: [0.23.5]
  py-proj:
    version: [1.9.5.1.1]      # Normal released version 1.9.5.1 is buggy
  py-matplotlib:
    variants: +gui +ipython
  py-numpy:
    variants: +blas +lapack

  ibmisc:
    version: [develop]
    variants: +python +netcdf
  icebin:
```

(continues on next page)

(continued from previous page)

```

version: [develop]
variants: +gridgen +python ~coupler ~pism

# Running without dynamic ice
modele:
  version: [landice]

# # Running with dynamic ice
# modele:
#   version: [glint2]
#   variants: [+couler +pism]

pism:
  version: [glint2]
glint2:
  version: [glint2]
  variants: +coupler +pism

everytrace:
  version: [develop]
eigen:
  variants: ~suitesparse
netcdf:
  variants: +mpi

# Recommended for security reasons
# Do not install OpenSSL as non-root user.
openssl:
  paths:
    openssl@system: /usr
  version: [system]
  buildable: False

# Recommended, unless your system doesn't provide Qt4
qt:
  paths:
    qt@system: /usr
  version: [system]
  buildable: False

all:
  compiler: [gcc@4.9.3]
  providers:
    mpi: [openmpi]
    blas: [openblas]
    lapack: [openblas]

```

## 2.2.4 Install ModelE Environment

This installs all the prerequisites needed to run ModelE, along with basic tools to analyze its output.

```

spack install modele-utils
spack install --dependencies-only modele
spack install ncview

```

(continues on next page)

(continued from previous page)

```
spack install nco
spack install modele-control
```

## 2.2.5 Generate the Module Loads

Run the following script, to generate the file `~/env/modele`. This will load the modules you just created:

```
#!/bin/sh
#
# Generate commands to load the Spack environment

SPACKENV=$HOME/env/modele
FIND='spack module loads'

echo '#!/bin/sh -f' >$SPACKENV
echo '# ---- Machine generated; do not edit!' >>$SPACKENV
#echo 'module purge' >>$SPACKENV

# --- Modele Stuff
$FIND ncview >>$SPACKENV
$FIND nco >>$SPACKENV
$FIND modele-control >>$SPACKENV
$FIND modele-utils >>$SPACKENV
```

### NOTES:

1. Remember to include any bootstrapping modules you might need as well: for example, pre-existing compilers sometimes must be loaded to run anything built with them.
2. Depending on how your system's environment modules are configured, you might need to add `--prefix` to the `spack module loads` command. See `spack module loads --help`.

## 2.2.6 Update `.bashrc`

Add the following to your `.bashrc` file, modifying accordingly:

```
export SPACK_ROOT=$HOME/spack
. $SPACK_ROOT/share/spack/setup-env.sh
export MODULEPATH=$SPACK_ROOT/share/spack/modules:$MODULEPATH
export PATH=$PATH:$HOME/spack/bin
alias spack='nice spack'
export SPACK_DIRTY=
export LESS='-R'
source $HOME/env/modele
```

## CHAPTER 3

---

### Build and Run ModelE

---

Now that the ModelE environment has been installed, it is possible to begin downloading and running climate models. This section serves as a tutorial, not reference manual. Definitive usage for any ModelE-Control command may be obtained via `ectl <cmd> --help`. For example:

```
$ ectl setup help
usage: ectl setup [-h] [--ectl ECTL] [--rundeck RUNDECK] [--src SRC]
                  [--pkgbuild] [--rebuild] [--jobs JOBS]
                  run

positional arguments:
  run                  Directory of run to setup

optional arguments:
  -h, --help            show this help message and exit
  --ectl ECTL           Root of ectl tree: ectl/runs, ectl/builds, ectl/pkgs
  --rundeck RUNDECK, -rd RUNDECK
                        Rundeck to use in setup
  --src SRC, -s SRC     Top-level directory of ModelE source
  --pkgbuild            Name package dir after build dir.
  --rebuild            Rebuild the package, even if it seems to be fine.
  --jobs JOBS, -j JOBS Number of cores to use when building.
```

### 3.1 Setup the ModelE Root

Begin by setting up a ModelE **root directory**; this must be an ancestor of all your run directories. It is marked as a root directory by the presence of a file named `ectl.conf`. For example:

```
$ mkdir ~/exp      # The root directory
$ echo >~/exp/ectl.conf
```

**Notes:**

1. Run directories need to be held within the root directory, but not necessarily as direct children. For example, the following directory structure is common:

```
exp/          # Root directory
  experiment1/
    run1/      # Run 1 of experiment 1
    run2/      # Run 2 of experiment 1
  experiment2/
    run1/      # Run 1 of experiment 2
    run2/      # Run 2 of experiment 2
```

2. There are no restrictions on what can go inside the root in addition to ModelE runs. Typically, they may contain pre-processing and post-processing code, graphs, ModelE source directories — anything needed by the user while building an experiment.
3. The user may have more than one root; although there is rarely a need to do so.
4. Any existing directory may be turned into a root. For example:

```
$ echo >~/ectl.conf  # Turn ~ into a root
```

## 3.2 Download ModelE Source

Once a root has been set up, the user must find or download a ModelE source directory. This directory can be anywhere on the filesystem, it does not have to live within a root. Source directories may be shared by multiple run directories. ModelE source is typically obtained by downloading from Simplex, or some other Git repository:

```
$ cd ~/exp      # Root directory
$ git clone simplex.giss.nasa.gov:/giss/gitrepo/modelE.git -b <branch>
```

### Notes:

1. <branch> is the ModelE branch you wish to use: master, develop, landice, cmake, etc.
2. The cmake build *must* be enabled on the branch you choose. You can tell if it is on your branch by looking for a file CMakeLists.txt or modele-control.pyar at the top level. So far, CMake is enabled on the branches master, landice and efischer/ec2.

If CMake is not enabled for your branch, copy the modele-control.pyar file from the master branch, and check it into your branch. Or merge from master.

## 3.3 Setup ModelE Source

From the ModelE download directory, type the following:

```
$ cd ~/exp/modelE
$ spack uninstall -ay modele@local; spack setup modele@local
```

This finds all of ModelE’s dependencies and creates a file spconfig.py, which is used in the build process to configure ModelE’s dependencies for your system. Alternately, you can copy spconfig.py from another working ModelE source directory.



### 3.3.1 Quick Setup: NCCS Discover

If you are running on NCCS Discover, you do not need to run Spack. Simplified instructions are:

```
$ cd ~/exp/modelE
$ ln -s $EHOME/env/modelE-spconfig.py spconfig.py
```

## 3.4 Create a Run

It is now possible to create a ModelE run directory. ModelE-Control needs to know which source directory and rundeck you wish to use for this run, as well as the name of the run directory you are creating. For example, suppose you wish to create a run directory called myrun:

```
$ cd ~/exp
$ ectl setup myrun --src ~/exp/modelE --rundeck ~/exp/modelE/templates/E4F40.R
```

This will do the following:

1. Create your run directory. Run directories may be created anywhere that is a sub-directory of the ModelE-Control root.
2. Link input files into the run directory, downloading any missing input files.
3. Record your choices of source directory and run directory; these will be saved as symbolic links called `src` and `upstream.R` inside your run directory. For example:

```
$ ls myrun
src -> ../../../../home/rpfische/f15/modelE
upstream.R -> ../e4f40.R
```

4. Create a build directory, where the source code for ModelE will be built. It will be created in a subdirectory `builds` of the ModelE-Control. In this case:

```
build -> ../builds/768603dc2b58f45a96b72c5839d79dbd
```

Note that the build directory is named by a random-looking hash. This hash is generated based on the ModelE source directory and the contents of your chosen rundeck; more on this later.

5. Use CMake to generate a build, linked up to the proper dependencies. This is done by running the `spconfig.py` script generated above by Spack:

```
-- CMAKE_INSTALL_RPATH /gpfs/dnb53/rpfische/exp/pkgs/
-- 1e35f5f359ecbb675e04a1c75f9ee260/lib
-- Found MPI_C: /usr/local/other/SLES11.3/openmpi/1.10.1/gcc-5.3/lib/libmpi.so
...
-- *****
-- ***** PROJECT: ModelE *****
-- Architecture: x86_64
-- System:      Linux
-- MODELERC:
-- COMPILER:    GNU 5.3.0
-- RUNSRC:
-- RUN:         /gpfs/dnb53/rpfische/exp/e4f40.R
-- MPI:         YES
-- WITH_PFUNIT:
-- *****
```

(continues on next page)

(continued from previous page)

```
-- Configuring done
-- Generating done
-- Build files have been written to: ~/exp/builds/9b3ea947a57318e1e33018503c16b82d
```

#### 6. Use make to build ModelE with the CMake-generated build:

```
[ 0%] Generating landice/ExportConstants.F90
[ 1%] Generating shared/RunTimeControls_mod.F90
[ 2%] Generating shared/Attributes.F90
[ 2%] Generating Ent/ent_mod.f
[ 3%] Generating shared/AttributeHashMap.F90, shared/AbstractTimeStamp.F90,
↳shared/CalendarDate.F90
[ 3%] Generating shared/AttributeDictionary.F90
Writing ../landice/ExportConstants.F90
Reading /home/rpfische/fl15/modelE/model/shared/Constants_mod.F90
Reading /home/rpfische/fl15/modelE/model/SEAICE.f
Scanning dependencies of target modele
[ 4%] Building Fortran object model/CMakeFiles/modele.dir/landice/DebugType.F90.o
...
[ 96%] Building Fortran object model/CMakeFiles/modele.dir/SURFACE.f.o
[ 97%] Building Fortran object model/CMakeFiles/modele.dir/STRAT_DIAG.f.o
[ 98%] Building Fortran object model/CMakeFiles/modele.dir/RAD_DRV.f.o
[ 98%] Linking Fortran shared library libmodele.so
[ 98%] Built target modele
Scanning dependencies of target modelexe
[ 99%] Building Fortran object model/CMakeFiles/modelexe.dir/main.F90.o
[100%] Linking Fortran executable modelexe
[100%] Built target modelexe
```

#### 7. Create a package directory, where the executable for this run will live. It will be created in a subdirectory pkgs of the ModelE-Control. In this case:

```
pkg -> ../pkgs/1e35f5f359ecbb675e04a1c75f9ee260
```

#### 8. Install the built ModelE binaries into the package directory:

```
Install the project...
-- Install configuration: "Release"
-- Installing: ../lib/libmodele.so
-- Set runtime path of "../lib/libmodele.so" to ...
-- Installing: ../bin/modelexe
-- Set runtime path of "../bin/modelexe" to ...
```

## 3.5 Start the Run

To start a run, for example, to run with two processors:

```
$ ectl run ~/exp/test -np 2
```

Note that this command works from any directory. You could just as well have typed:

```
$ cd ~/exp
$ ectl run test
```

or even:

```
$ cd ~/exptest
$ ectl run
```

Before launching ModelE, this command will generate the ModelE *I* file based on your run's *rundeck.R* file. This ensure that any parameter changes made to *rundeck.R* will be reflected in *I*. The user should *never* have to edit the *I* file directly.

This will start the run in the background and return to your shell prompt. The run will continue until it ends by itself or is stopped; logging out will NOT stop the run. After starting the run, ModelE-Control shows run status:

```
$ mpirun -timestamp-output -output-filename /gpfs/dnb53/rpfische/exp/test/log/q -np 2 --report-pid /gpfs/dnb53/rpfische/exp/test/model.e.pid /gpfs/dnb53/rpfische/exp/test/pkg/bin/model.exe -cold-restart -i I
nohup: ignoring input and appending output to `nohup.out'
===== test
status:  RUNNING
run:      /gpfs/dnb53/rpfische/exp/test
rundeck:  /gpfs/dnb53/rpfische/exp/e4f40.R
src:      /gpfs/dnb53/rpfische/fl15/model.e
build:    /gpfs/dnb53/rpfische/exp/builds/768603dc2b58f45a96b72c5839d79dbd
pkg:      /gpfs/dnb53/rpfische/exp/pkgs/1e35f5f359ecbb675e04a1c75f9ee260
launcher = mpi
pidfile = /gpfs/dnb53/rpfische/exp/test/model.e.pid
mpi_cmd = mpirun -timestamp-output -output-filename /gpfs/dnb53/rpfische/exp/test/log/q -np 2 --report-pid /gpfs/dnb53/rpfische/exp/test/model.e.pid
model.e_cmd = /gpfs/dnb53/rpfische/exp/test/pkg/bin/model.exe -cold-restart -i I
cwd = /gpfs/dnb53/rpfische/exp/test
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
rpfische  436  7.9  0.0 4280812 4124 pts/9    Sl+   17:31   0:00 mpirun -timestamp-output -output-filename /gpfs/dnb53/rpfische/exp/test/log/q -np 2 --report-pid /gpfs/dnb53/rpfische/exp/test/model.e.pid /gpfs/dnb53/rpfische/exp/test/pkg/bin/model.exe -cold-restart -i I
rpfische  443 86.8  0.1 13635064 245040 pts/9    Dl    17:31   0:00 /gpfs/dnb53/rpfische/exp/test/pkg/bin/model.exe -cold-restart -i I
rpfische  445 92.2  0.1 13624436 242348 pts/9    Rl    17:31   0:00 /gpfs/dnb53/rpfische/exp/test/pkg/bin/model.exe -cold-restart -i I
```

## 3.6 View the Log

The ModelE STDOUT/STDERR log file(s) are written into the directory `myrun/log`, and are named by MPI rank:

```
$ ls -l log
total 960
-rw-r----- 1 rpfische s1001 599042 Aug 28 17:32 q.1.0
-rw-r----- 1 rpfische s1001 329834 Aug 28 17:32 q.1.1
```

Output is separated by MPI rank to enhance scalability, and to avoid the occasional garbled output when two MPI ranks write output at the same time. Timestamps in the per-rank log files allow them to be combined into one file if desired.

While ModelE is running, a log file may be watched via:

```
$ ectl tail -f myrun
```

## 3.7 Manage the Run

After a run has been started, you can inspect the status of the run; for example:

```
$ ectl ps myrun
```

If you have many runs going at once, you can also inspect the status of them all together. For example:

```
$ ectl ps myrun1 myrun2
```

or to get the status of all the runs in your ModelE-Control root:

```
$ cd ~/exp
$ ectl ps
```

In any case, the status will tell the current model date/time, and whether the simulation is currently running. For example, after a simulation has terminated, `ectl ps` looks like:

```
===== test
status:  STOPPED
itime =      16033 timestamp = 1949-12-01T00:00
fort.1.nc: 1949-12-01 00:00:00
fort.2.nc: 1949-12-01 01:00:00
run:      /gpfs/dnb53/rpfische/exp/test
rundeck:  /gpfs/dnb53/rpfische/exp/e4f40.R
src:      /gpfs/dnb53/rpfische/fl15/modelE
build:    /gpfs/dnb53/rpfische/exp/builds/768603dc2b58f45a96b72c5839d79dbd
pkg:      /gpfs/dnb53/rpfische/exp/pkgs/1e35f5f359ecbb675e04a1c75f9ee260
launcher = mpi
pidfile = /gpfs/dnb53/rpfische/exp/test/modelE.pid
mpi_cmd = mpirun -timestamp-output -output-filename /gpfs/dnb53/rpfische/exp/test/
↪log/q -np 2 --report-pid /gpfs/dnb53/rpfische/exp/test/modelE.pid
modelE_cmd = /gpfs/dnb53/rpfische/exp/test/pkg/bin/modelexe -cold-restart -i I
cwd = /gpfs/dnb53/rpfische/exp/test
<No Running Processes>
```

## 3.8 Stop the Run

In order to stop a run:

```
$ ectl stop myrun
```

This will do a “soft stop” by requesting ModelE to terminate. It is also possible to do a “hard stop” that kills the ModelE process as expediently as possible:

```
$ ectl stop -f myrun
```

Once the stop process is complete, `ectl ps` output should reflect that.

## 3.9 Post-Mortem

Once a ModelE run has stopped, it is possible to determine how it stopped, using Everytrace:

```
$ ectl trace myrun

===== Resolving Everytrace-enabled binaries:
  /gpfs/dnb53/rpfische/exp/pkgs/1e35f5f359ecbb675e04a1c75f9ee260/lib/libmodele.so
ref_addr_lib 495072 /gpfs/dnb53/rpfische/exp/pkgs/1e35f5f359ecbb675e04a1c75f9ee260/
↪ lib/libmodele.so
===== q.1.0
Exiting with return code: 13
0x7FFEFB7804C7
0x7FFEFBA860D6
0x7FFEFBA8612D
/home/rpfische/f15/modelE/model/MODELE.f:448
/home/rpfische/f15/modelE/model/MODELE_DRV.f:28
0x400A57
0x7FFEFAD35C35
===== q.1.1
Exiting with return code: 13
0x7FFEFB7804C7
0x7FFEFBA860D6
0x7FFEFBA8612D
/home/rpfische/f15/modelE/model/MODELE.f:448
/home/rpfische/f15/modelE/model/MODELE_DRV.f:28
0x400A57
0x7FFEFAD35C35
```

Everytrace provides a stacktrace, with filenames and line numbers, of how ModelE stopped on each MPI rank. In this case, ModelE terminated on line 448 of `MODELE.f`, which is normal termination:

```
CALL stop_model('Terminated normally (reached maximum time)',13)
```

In this case, normal termination can also be confirmed by inspecting the log files.

---

**Note:** The Everytrace feature is currently enabled only on the `landice` branch.

---



### 4.1 Elevation Classes

ModelE can run with *elevation classes* for land ice. In this mode, the ice surface model is run at multiple fixed elevations for each atmosphere gridcell, also known as the *elevation grid*.

In order to run with elevation classes, ModelE needs appropriate matrices to regrid from the elevation grid to atmosphere grid (going the other way is assumed to be trivial). These are computed using the *IceBin* regridding / coupler library; although when not running coupled, only the regridding portions are used. The following changes are required to ModelE input files:

- The NetCDF variable  $fhc(nhc, jm, im)$  in the *TOPO* file, where  $nhc$  is the number of elevation classes.  $fhc(ihc, j, i)$  is extracted from the *EvI* sparse matrix generated by *IceBin*:  $fhc(ihc, j, i)$  must be set to the value of  $AvE$  at with coordinates  $((j, i), (ihc, j, i))$ ; in other words, the contribution of elevation grid cell  $(ihc, j, i)$  to atmosphere grid cell  $(j, i)$ .
- The NetCDF variable  $elevE(nhc, jm, im)$  in the *TOPO* file must provide the elevation for each elevation class. Typically, these will be set at fixed elevations (eg. every 100 meters) and be the same for all atmosphere gridcells. The need to be whatever elevations were used in *IceBin*.
- If using the classic 3m snow/firn model:
  - The variables *snowli* and *tlandi* in the *GIC* file must be extended with a new  $nhc$  dimension, and set appropriately. If you don't know how to set them, just set all elevation classes the same for spinup.

```
double snowli(nhc, jm, im) ; double tlandi(nhc, jm, im, d2) ;
```

- If using the Stieglitz snow/firn model, Stieglitz-appropriate variables are required: .. code-block:: console
 

```
double dz(nhc, jm, im, nlice) ; double wsn(nhc, jm, im, nlice) ; double hsn(nhc, jm, im, nlice) ; double
tsn(nhc, jm, im, nlice) ;
```

## 4.2 Continuing a Run

If a run directory has stopped running, it may be restarted where it left off with *ectl run*. For example:

```
$ ectl run myrun --time 12:00:00 -np 28
```

---

**Note:** Command-line parameters related to HOW to run this job must be repeated, since they might be different from the last run: `--launcher`, `--ntasks`, `--time`.

---

The above command rewrites the I file from your edited `rundeck.R`. This makes it easy to change rundeck parameters and restart arun. However, any command-line modifications to start/end time will be lost. If this behavior is not desired, you can either:

1. Specify the end time again on the command-line. For example to end in 1960:

```
$ ectl run myrun --timespan ,1960-01-01 --time 12:00:00 -np 28
```

2. Put the end time in your `rundeck.R`, eliminating the need to specify it on the command line.
3. Continue the run with the `--resume` option. This will use the I file from the last run, rather than the rundeck. Using this option, it is not possible to change rundeck parameters:

```
$ ectl run myrun --resume --time 12:00:00 -np 28
```

## 4.3 Restarting from fort.X.nc or an .rsf file

You can restart a run from the `fort.1.nc`, `fort.2.nc` or a restart (`.rsf`) file using the `--restart-file` option. Examples include:

```
$ ectl run myrun --restart-file myrun/fort.1.nc --time 12:00:00 -np 28
$ ectl run myrun --restart-file myrun/1MAR1957.rsfmtmyrun.R.nc --time 12:00:00
↪ -np 28
```

The `fort.X.nc` files will be overwritten. If you want to restart from one of them while ensuring your restart file is not overwritten, copy `fort.X.nc` to a different name. For example:

```
$ cd myrun
$ cp fort.1.nc myrestart.nc
$ ectl run . --restart-file myrestart.nc --time 12:00:00 -np 28
```

## 4.4 Restarting by Date

It is also possible to restart from an `.rsf` file by specifying a date. For example:

```
$ ectl run myrun --restart-date 1957-03-01 --time 12:00:00 -np 28
```

This command will find the restart file for March 1957 and restart from it. If there is no restart file for the date you choose, ModelE-Control will restart from the most recent “.rsf” file less than the date. For example, the following will produce the same result when used with monthly restart files:



```
$ ectl run myrun --restart-date 1957-03-17 --time 12:00:00 -np 28
```

## 4.5 Keepalive

When you start a run, ModelE lists it in a `ectl/keepalive.txt` file inside your ModelE-Control root. The command `ectl keepalive` will read that file and continue any runs that have stopped because they have reached their SLURM time limit. To use it:

```
$ ectl keepalive <ectl-root> --time 12:00:00 -np 28
```

---

### Note:

1. `<ectl-root>` could be the ModelE-Control root, or any subdirectory thereof.
  2. As with `ectl run`, you need to re-specify the arguments `--launcher`, `--ntasks` and `--time`.
  3. This command is intended to run periodically — say, every 5 minutes, from a cron job.
  4. `ectl keepalive` will not restart jobs that have terminated on their own, or crashed, or were forcibly evicted from your cluster.
- 

## 4.6 Create a Rundeck

Although ModelE-Control can work directly out of the `templates` directory, it can also assemble rundecks for further manual editing. This is done with *ectl flatten*. Rundecks may be created in any directory on the filesystem:

```
$ cd ~/exp
$ ectl flatten modelE/templates/E4F40.R e4f40.R
```

## 4.7 One Root per User

Alternately, users may choose to have only one root, presumably in the user's home directory. ModelE-Control then manages only one `builds` and `pkgs` directories for the entire user. This simplifies management in some ways, but it slows down certain `ectl` operations (`ps`, `purge`).



## CHAPTER 5

---

### Elevation Classes

---



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`