SANDIA REPORT

SAND2017-3771
Unlimited Release
Printed April 13, 2017

Guide to Using Sierra

Ryan P. Shaw, Anthony M. Agelastos, Joel D. Miller

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2017-3771
Unlimited Release
Printed April 13, 2017

Guide to Using Sierra

Ryan P. Shaw, Anthony M. Agelastos, Joel D. Miller
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185
sierra-help@sandia.gov

Abstract

Sierra is an engineering mechanics simulation code suite supporting the Nation's Nuclear Weapons mission as well as other customers. It has explicit ties to Sandia National Labs' workflow, including geometry and meshing, design and optimization, and visualization. Distinguishing strengths include "application aware" development, scalability, SQA and V&V, multiple scales, and multi-physics coupling. This document is intended to help new and existing users of Sierra as a user manual and troubleshooting guide.

Acknowledgments

This document's authors acknowledge Mike Glass for the original idea of a Sierra troubleshooting guide and Justin Lamb and others for proofreading this manual.

The following teams were very helpful in content for the following sections:

Compsim Devops Code versions (Chapter 6)

Sierra Solid Mechanics Restart (Section 5.4)

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ASC	NNSA's Advanced Simulation and Computing Program
ATS	Advanced Technology Systems
BASH	Bourne Again SHell
CEE	Common Engineering Environment
CSH	C SHell
DOE	U.S. Department of Energy
EOS	End-of-Sprint
FCT	Feature Coverage Tool
HPC	High-performance Computing
I/O	Input/Output
JIT	Just-in-Time
LDRD	Laboratory-directed Research & Development
MPI	Message Passing Interface
MPMD	Multiple Program Multiple Data
NNSA	DOE's National Nuclear Security Administration
NTK	Need-to-Know
OST	Object Storage Target
OUO	Official Use Only
PO	Product Owner
POSIX	Portable Operating System Interface
SAW	Sandia Analysis Workbench
SCN	Sandia Classified Network

SEACAS	Sandia National Laboratories Engineering Analysis Code Access System
SNL	Sandia National Laboratories
SQA	Software Quality Assurance
SRN	Sandia Restricted Network
V&V	Verification & Validation
VOTD	Version of the Day
WCID	Workload Characterization ID
WFO	Work For Others

Other Terms

CompSim	Sierra Computational Simulation website
DevOps	Development Operations
`master`	Version of the Day
STDOUT	Standard Output

Major Sierra Applications

Adagio	Sierra/SolidMechanics or Sierra/SM
Aero	Sierra/Aero
Aria	Sierra/Thermal Multiphysics
Fuego	Sierra/Low Mach Fluids
Presto	Sierra/SolidMechanics or Sierra/SM (only in ITAR Sierra)
Salinas	Sierra/Structural Dynamics or Sierra/SD
STK	Sierra ToolKit

Chapter 1

Introduction

This document is a troubleshooting guide for common problems encountered by Sierra analysts. Note that while much of this document can be applied to all Sierra users, some sections are only applicable to internal Sandia users of Sierra. Items and Sections applicable only to Sandia users are marked by the Sandia National Labs Thunderbird logo:

A list of chapters and their descriptions is given below.

Chapter 2. Frequently Asked Questions
This chapter contains several frequently asked questions and issues that Sierra Help has dealt with in the past. It is recommended to look here to determine whether your issue has been encountered and solved in the past.

Chapter 3. Before Asking for Help
This is a good chapter to go through before contacting Sierra user support or Sierra developers since it helps the reader prepare their help request along with providing guidance on common problems.

Chapter 4. User Environment
This chapter provides the steps and information necessary to initially use Sierra, which is helpful for those new to Sierra, and to set up their environment to properly utilize Sierra on the HPC and CEE resources, which is helpful to all users.

Chapter 5. Running Sierra
This chapter provides background to users regarding how to execute Sierra on the command line as well as within job scripts including advanced topics such as building user subroutines, initiating checkpoint/restart files, utilizing memory effectively, and code coupling.

Chapter 6. Versioning
This chapter outlines Sierra's versioning system and details useful information regard-

ing the number of versions that are officially supported, how to use different versions, and the considerations to make before using certain versions of Sierra.

Chapter 7. Development Process

The chapter gives information about the Sierra development process, which is especially useful to know when requesting new features for one of the Sierra application codes.

There are some additional references within the appendices as well. Appendix A contains a document convention guide to help interpret the different styles in this manual. Appendix B contains information about how to obtain and build this document. Appendix C contains an example Sierra input file along with tips for how to properly format it. Appendix D contains example `.bashrc` and `.bash_profile` files. Appendix E lists an equivalent example `.cshrc` file.

Chapter 2

Frequently Asked Questions

What does this command line error mean: “`Sierra modules loaded, but no installed <version> applications were found to put in PATH.`”?

This error occurs when a Sierra module, but the applications specific to that version of Sierra do not exist on the system. This most often occurs when a user loads the `sierra/master` module, since the master builds may not happen daily due to development volatility (see Section 6.3). If this error occurs for some other version of the code, there is likely some installation issue and you should alert Sierra Help so that the issue can be resolved.

I cannot load any `sierra` modules.

This typically occurs for two different reasons:

1. you do not have the correct accounts in WebCARS to allow loading the module. See Section 4.1 to acquire the proper accounts.
2. The terminal environment is stale - refresh the environment by opening a new terminal or logging out and back in (see Section 4.5).

Help debugging error: “`error while loading shared libraries: lib*.so: cannot open shared object file.`”

This type of error occurs when the user environment is incorrect (i.e., incompatible with the Sierra executable’s build environment), which may arise for several reasons:

1. Trying to run in a stale environment (see Section 4.5)
2. Attempting to run with a user subroutine that was compiled with a different version of the code (see Section 5.5)
3. Loading many different modules concurrently in the same terminal (see Section 4.4.1)

Where are user manuals located in a Sierra distribution?

Please see Section 3.1.

I get an error at the end of my run about the Feature Coverage Tool.

The Feature Coverage Tool (FCT) is an application created by the Verification & Validation team to help determine the level of testing for the features in your model. This tool is automatically run after all successful runs, but there are sometimes issues with the FCT. If you see errors related to this, typically your run was successful and you can still post-process the results. Please let Sierra Help know about these FCT issues so that they can be addressed in a timely manner.

How can I use the `sierra` script to only perform the pre- or post-processing step?

The `sierra` script (Section 5.2) provides the command line options `--pre` and `--post`, which will perform the pre- or post-processing steps, respectively, instead of performing all three pre, run, and post steps. For more information about these steps, see Section 5.3.

How can I run Sierra using the `sierra` script on an HPC interactive node without submitting to the queue?

It is sometimes desirable to request an HPC interactive node to perform quick debugging on HPC platforms. For example, to request 1 interactive node on SNL HPC platforms for 4 hours, one might do the following:

```
$ salloc -N1 --time=4:00:00 --account=your_WC_ID
```

After obtaining access to the interactive node, one can then use the `sierra` script without submitting to the HPC queue as follows:

```
$ module load sierra
$ export NO_SUBMIT=true
$ sierra --np 16 adagio -i input.i
```

See Section 5.2 for more information on the `sierra` script and usage.

How does one overcome this error: `ACCOUNT: Setting ACCOUNT is required for submitting to queue.`

This error occurs on HPC platforms when a valid WCID is not specified (see Section 4.2). To overcome this, follow the steps in the aforementioned section, and if you already have a valid WCID, then pass the `--account` option to the `sierra` script. If you only have one WCID, then another option is to set the `ACCOUNT` environmental variable to your WCID, which will automatically set the WCID for that environment.

How can I tell which version of Sierra I am using?

There are several different methods for determining the version of Sierra in your current environment, listed in order from most to least effective:

```
$ which <SIERRA_APPLICATION>
```

```
$ module list      # See Section 4.4
```

```
$ <SIERRA_APPLICATION> --version
```

How many versions of Sierra are supported?

Sierra supports two major releases at any time, as well as two End-of-Sprint snapshots. For more information regarding Sierra's versioning system, see Chapter 6.

Chapter 3

Before Asking for Help

When you first encounter an issue, there are some resources you can investigate before submitting a help ticket.

3.1 Documentation

Invariably, many syntax questions can be solved by a quick scan of the available documentation. The central repository of all available Sierra documentation can be found on the Sierra Computational Simulation (CompSim) website, which can be accessed at the following URL (for Sandia users only):

http://compsim.sandia.gov/compsim/

To find documentation on this site, navigate to "Support & Services" → "Documentation." On this site, one can find documentation for current and previous releases as well as `master` (a.k.a. Version of the Day) documentation. Master documentation is located under the "Pre-Release" section.

For external Sierra users, manuals can be found at the following location in each release distribution:

```
X.YY.Z/release-manuals/Docs/Sierra/X.YY/GeneralRelease/
```

Users who navigate to this directory may open up individual documents specific to their model, or they may open the index.html file in that location to see a tree of all Sierra documents.

The following is a representative list of documentation that one might find at the aforementioned documentation locations:

- Theory manuals
- User manuals
- Command summaries

- Release notes
- Training/Tutorials
- Verification problems
- Example problems
- Miscellaneous reports

Other items may be found on the CompSim website, including "Getting Started" documentation, application code summaries, and contact information, among other useful information.

Much of the documentation that is contained in the "Version of the Day" section of CompSim is kept up-to-date via automated processes that are checked daily to ensure correct documentation. Sierra Help has also placed Version of the Day (VOTD) documentation on the CEE LAN (both on the unclassified and classified networks) and on Sandia's high-performance computing (HPC) platforms:

```
/projects/sierra/doc
```

3.2 Additional Non-Sierra Resources

Sierra users will inevitably utilize Sierra on a variety of computational platforms and will need to perform advanced platform-specific operations. When issues arise, one can turn to several resources which are described below.

3.2.1 computing.sandia.gov

The Sandia website for Scientific, Engineering, and High Performance Computing can be found at the following URL `https://computing.sandia.gov/` This website contains much useful information, including the following:

- Platform specifications
- Application information
- Queuing and filesystem pointers
- Data storage options
- Current HPC news and notifications
- System policies

- Example run scripts

This website is a good place to start when beginning high performance computing or when doing something unfamiliar on these systems.

3.2.2 <machine>-help

For issues that seem to be related to the current system being utilized rather than those arising directly from a Sierra application, send an e-mail to that platform's help e-mail address. For example, for issues on Sky Bridge, one may email skybridge-help@sandia.gov. The personnel who man these e-mail groups will be better equipped to help you with issues related to platforms, filesystems, etc. If one is unsure of the specific platform or HPC issue, one may simply send an e-mail to hpc-help@sandia.gov. The members of Sierra Help are also happy to relay such messages to the appropriate team.

3.2.3 Post-processing Help

Many computational models run with Sierra are complex and require complex post-processing. As such, issues may arise when using such post-processing tools as ParaView [1] and EnSight [2]. Each of these software tools has a corresponding help e-mail: paraview-help@sandia.gov and ensight-help@sandia.gov, respectively.

3.3 Information to Give Sierra Help

At some point, you will have exhausted all possible resources and will put your question to Sierra Help, who can be reached at the following e-mail address: sierra-help@sandia.gov An overview of the general Sierra Help process can be seen in Figure 3.1. An understanding of this process can be useful in understanding support ticket progress and how best to interface with members of Sierra Help.

In order to quickly and efficiently solve your problem, certain information is required and will be explained below:

- Code version
- User environment
- Number of processors
- Files required to replicate issue
- Funding program

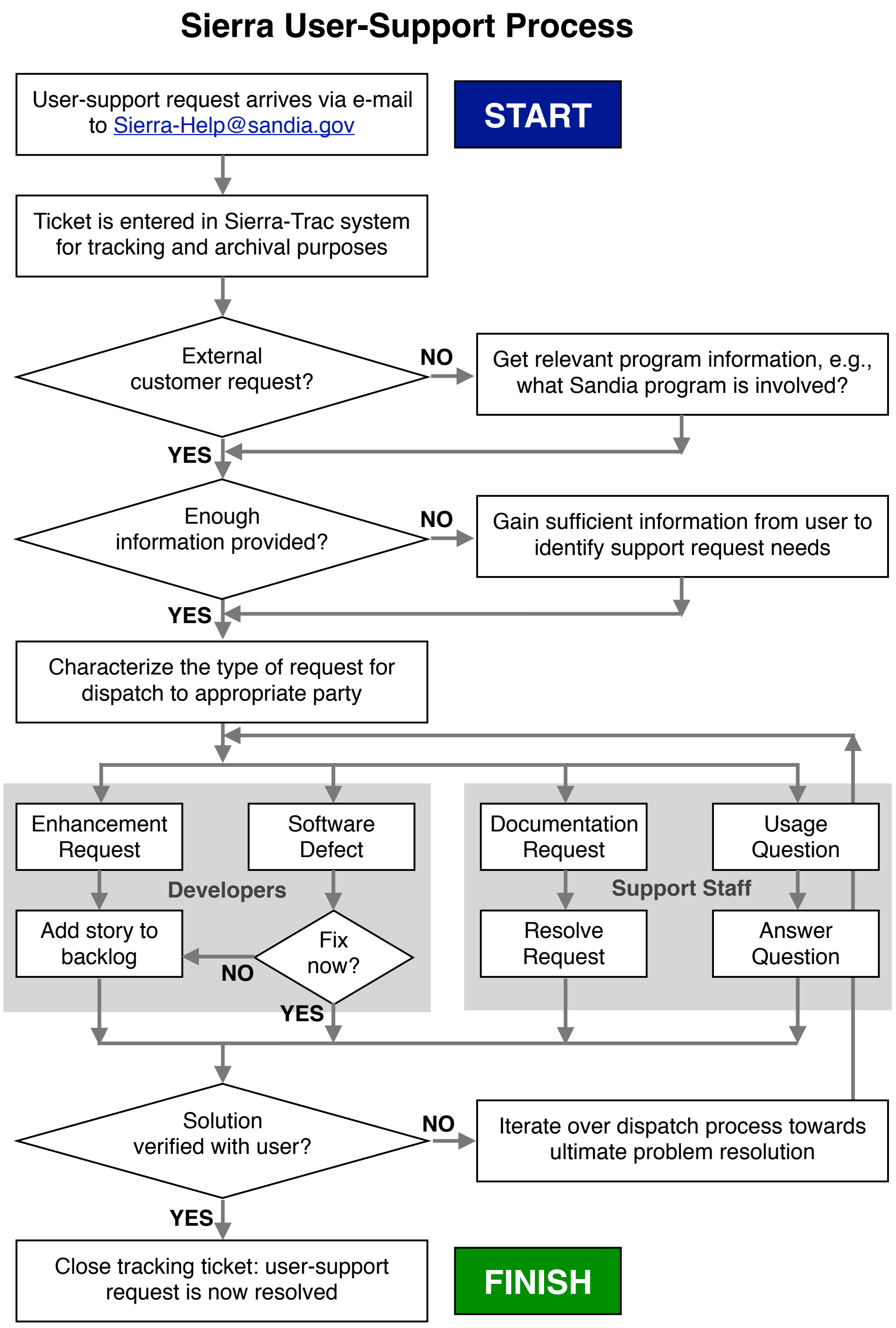

Sierra User-Support Process
User-support request arrives via e-mail to Sierra-Help@sandia.gov
START
Ticket is entered in Sierra-Trac system for tracking and archival purposes
External customer request?
NO
Get relevant program information, e.g., what Sandia program is involved?
YES
Enough information provided?
NO
Gain sufficient information from user to identify support request needs
YES
Characterize the type of request for dispatch to appropriate party
Enhancement Request
Software Defect
Documentation Request
Usage Question
Developers
Support Staff
Add story to backlog
Fix now?
NO
YES
Resolve Request
Answer Question
Solution verified with user?
NO
Iterate over dispatch process towards ultimate problem resolution
YES
Close tracking ticket: user-support request is now resolved
FINISH

Figure 3.1. Diagram of overall Sierra Help process.

3.3.1 Code Version

Sierra supports the use of many different code versions, as explained in Chapter 6. As such, each problem may perform differently when using different code versions, and the problem you are encountering may not even occur using a different code version. In order for us to troubleshoot the problem, we must use the same version in order to replicate and solve the problem or suggest the use of an improved version.

3.3.2 Running Environment

"Running environment" here refers to the environment in which you are running Sierra when you encounter a problem. This includes such information as follows:

- Machine (e.g., personal blade/workstation, specific HPC platform)
- Loaded modules (See Section 4.4)
- Non-default `.*rc` files (See Section 4.3)
- Custom run scripts

This information will allow Sierra Help to troubleshoot the environment, if that is the issue, or to replicate the environment for future investigation.

3.3.3 Number of Processors

Some issues that arise can be dependent upon whether a model is run in serial (single processor) or parallel (multiple processors) and even the number of processors used. As such, this information is very important in determining the source of the issue, such as code bugs, memory usage, and mesh decomposition.

3.3.4 Files

In the event that an issue arises when running a specific model, the best thing to do is to either attach all files necessary to run the model, or point members of Sierra Help to the location of the model files (see Section 3.4 for information on proper file permissions). Some HPC platforms support the use of `give` and `take` utilities, and use of these tools should be communicated with Sierra Help so they can act accordingly. Please also let us know of any special considerations, e.g., Official Use Only (OUO) requirements.

If the model under question is classified, **do not send classified files to sierra-help@sandia.gov**, as this is on an unclassified domain. Also realize that debugging classified problems often takes much longer due to limited resources. If at all possible, please make a simplified unclassified problem that exhibits the same issue so that time to solution can be shortened.

3.3.5 Funding Program

For tracking purposes, please identify the name of the Sandia program that is associated with your request. This information helps with user support cost recovery and metrics tracking. Some examples of program names include the following:

- ASC
- B61
- B83
- LDRD
- W88
- WFO

3.3.6 Additional Considerations

If there are any any additional pieces of information that could be relevant to the problem, please include this information. Sierra Help will open a ticket in order to resolve the issue quickly and efficiently.

3.4 Permissions

A common problem when sharing files is that permissions are not properly set, which then requires more communication to fix permissions. Due to the importance of security in the workplace, a good knowledge of UNIX file permissions will allow for better security. Here is an example of a listing of a representative set of files:

```
$ ls -l
total 8
-rwxr-x--- 1 jdoe jdoe             44267 Sep 14 22:49 binary
drwxr-xr-x 2 jdoe jdoe              4096 Aug  3 15:23 dir1
drwxr-s--- 2 jdoe wg-sierra-users   4096 Jan 29 01:33 dir2
```

```
-rw-r--r-- 1 jdoe jdoe            153402 Jun  5  2012 file1
-rw------- 1 jdoe jdoe               498 Dec 12 13:01 file2
```

The first ten characters in lines 3 through 7 denote different permissions-related aspects, or bits, of an object. The first entry classifies the object: `d` for a directory, `l` for a link, or `-` for a file. The other nine entries are grouped into triplets: user, group, and other, where the user denotes the owner of the object. In each triplet, the entries denote read, write, and execute privileges, respectively.

Let us now look at the five example objects listed above:

`binary`	The owner, `jdoe` (column 4), has read (`r`), write (`w`), and execute (`x`) privileges on this file, and members of group `jdoe` (column 5) have read (`r`) and execute (`x`) privileges.
`dir1`	This directory has read (`r`), write (`w`), and execute (`x`) for user `jdoe`, read (`r`) and execute (`x`) privileges for members of group `jdoe`, and read (`r`) and execute (`x`) privileges for all other users.
`dir2`	This directory has read (`r`), write (`w`), and execute (`x`) for user `jdoe` and read (`r`) and execute (`x`) privileges for members of group `wg-sierra-users`. Additionally, because of the SGID (`s`) set on the group's executable permission, all files created within `dir2` (after the SGID is set) will inherit `wg-sierra-users` as the group.
`file1`	Everyone can read (`r`) this file, and user `jdoe` can also write to this file.
`file2`	Only user `jdoe` can read and write to this file - no one else can access it in any manner.

In order to manipulate permissions and groups, two important UNIX commands are required: `chmod` and `chgrp`. The `chmod` command serves to change permissions, while the `chgrp` command is used to to change the group membership of a file. For example, to give read privileges to all users and to propagate its execute status, if present (accomplished by `X`), one could issue either of the following commands:

```
$ chmod ugo+rX file
$ chmod a+rX file
```

These attributes each have an octal representation, as shown in Table 3.1. As an example, the following command gives read and write privileges to the user, read privileges to the group, and no privileges to others:

```
$ chmod 640 file
```

One important aspect of permissions is that *in order to allow an entity to enter a directory, the executable permissions must be enabled on the directory for that entity.* For example, to allow everyone to access a directory, use the following command:

Table 3.1. Octal bit representations for permissions triplets

#	Permission	`rwx`
7	full	111
6	read and write	110
5	read and execute	101
4	read only	100
3	write and execute	011
2	write only	010
1	execute only	001
0	none	000

```
$ chmod a+X dir
```

One who is not familiar with UNIX permissions may opt for the naïve method of completely opening permissions on all files, e.g.,

```
$ chmod -R 777 *   ### DO NOT DO THIS!! ###
```

While this does certainly allow someone quick access to files, it also has undesirable effects. First, giving everyone access to all files including write access, is generally discouraged due to its high level of insecurity and possibility of file corruption or loss. The above command also gives execute permissions to all objects, which is incorrect, as most files should not and cannot be executed. The above command should be avoided in the vast majority of circumstances.

One can circumvent future permissions issues with two additional commands. The first command is `umask`, which sets the default permissions for newly created files and directories in a terminal session. To give full privileges to the owner, read and execute permissions to the group, and no permissions to others, use this `umask` command:

```
$ umask 0027
```

The next command deals with the SGID, which was explained in the example above (`dir2`). To set the SGID on a directory, one can use the following command:

```
$ chmod g+s dir2
```

The above command ensures that all files and directories created within `dir2` will inherit the same group as `dir2`.

Members of Sierra Help have created a useful script that allows users to simplify the process of opening permissions to files. This script, `fixperm.sh`, is made available upon loading the `sierra` module (see Section 4.4). One may either specify certain directories and files, as follows:

```
$ fixperm.sh /path/to/directory file1 file2
```

or one may enter the directory in which the files are contained and issue the following command:

```
$ fixperm.sh .
```

There are other useful options for this script that can be found by invoking the script without any options, i.e.,

```
$ fixperm.sh
```

Chapter 4

User Environment

It is necessary to set up the user environment before using Sierra. For external users, this is best accomplished by contacting your system administrator and referring to Sections 4.3 and 4.5. For Sandia users, all of the following sections are applicable.

4.1 Necessary Accounts

For Sandia users, the first step to getting the user environment set up is to add Sierra-related accounts on WebCARS:

https://webcars.sandia.gov/

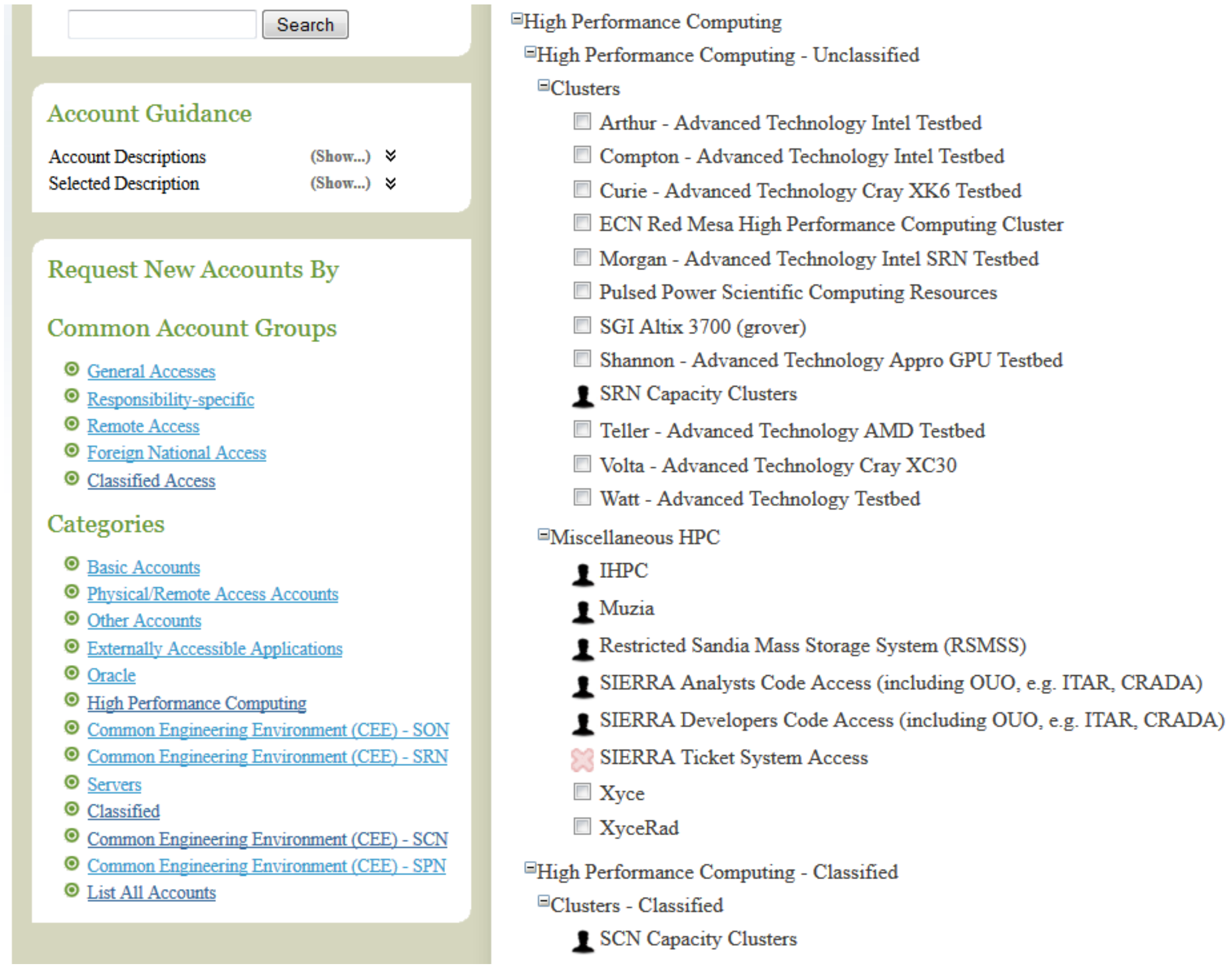
Once there, navigate to "High Performance Computing," located on the left of the home-page. Figure 4.1 shows the accounts available at this location, including Sierra-specific accounts. If one desires only to run Sierra, then the "SIERRA Analysts Code Access" account must be selected. If one desires access to the source code, then the "SIERRA Developers Code Access" should also be selected. In addition, to run Sierra on HPC platforms, the "SRN/SCN Capacity Clusters" accounts should be selected. This information is outlined on CompSim at the following URL:

http://compsim.sandia.gov/compsim/Support_Availability.html#access

WebCARS account approval depends on manager and sometimes other approval, which can take some time (typically around a day, with the exceptions of approver absence). After the accounts are approved, another day is required for the accounts to propagate to all SNL systems. In order for changes to take effect in your local user environment on a SNL system, you must log off and back on to that system.

4.2 WCID Process and Usage

A Workload Characterization ID (WCID) is required to run Sierra on one of SNL's HPC capacity platforms (e.g., TLCC2, Sky Bridge). Requests for new WCIDs should be made on



Search

Account Guidance

Account Descriptions (Show...)
Selected Description (Show...)

Request New Accounts By

Common Account Groups

- General Accesses
- Responsibility-specific
- Remote Access
- Foreign National Access
- Classified Access

Categories

- Basic Accounts
- Physical/Remote Access Accounts
- Other Accounts
- Externally Accessible Applications
- Oracle
- High Performance Computing
- Common Engineering Environment (CEE) - SON
- Common Engineering Environment (CEE) - SRN
- Servers
- Classified
- Common Engineering Environment (CEE) - SCN
- Common Engineering Environment (CEE) - SPN
- List All Accounts

⊟High Performance Computing
- ⊟High Performance Computing - Unclassified
  - ⊟Clusters
    - ☐ Arthur - Advanced Technology Intel Testbed
    - ☐ Compton - Advanced Technology Intel Testbed
    - ☐ Curie - Advanced Technology Cray XK6 Testbed
    - ☐ ECN Red Mesa High Performance Computing Cluster
    - ☐ Morgan - Advanced Technology Intel SRN Testbed
    - ☐ Pulsed Power Scientific Computing Resources
    - ☐ SGI Altix 3700 (grover)
    - ☐ Shannon - Advanced Technology Appro GPU Testbed
    - 👤 SRN Capacity Clusters
    - ☐ Teller - Advanced Technology AMD Testbed
    - ☐ Volta - Advanced Technology Cray XC30
    - ☐ Watt - Advanced Technology Testbed
  - ⊟Miscellaneous HPC
    - 👤 IHPC
    - 👤 Muzia
    - 👤 Restricted Sandia Mass Storage System (RSMSS)
    - 👤 SIERRA Analysts Code Access (including OUO, e.g. ITAR, CRADA)
    - 👤 SIERRA Developers Code Access (including OUO, e.g. ITAR, CRADA)
    - ✗ SIERRA Ticket System Access
    - ☐ Xyce
    - ☐ XyceRad
- ⊟High Performance Computing - Classified
  - ⊟Clusters - Classified
    - 👤 SCN Capacity Clusters

Figure 4.1. Example showing WebCARS high performance computing accounts, including those necessary to use Sierra

Sandia's WC Tool site:

`https://wctool.sandia.gov/wctool/`

Information related to the nature of the work must be submitted according to the directions on that site and may include the following:

- List of users (names and logins)
- Project/Task number
- Classification (SRN or SCN)
- Description of work

Any special requests can also be made by submitting an e-mail to wctool-help@sandia.gov.

Once a WCID is approved, users can check what their WC ID is by either using the `mywcid` or `sbatch` command on HPC platforms. More information about WC IDs can be found on computing.sandia.gov.

To simplify the usage of WCIDs on HPC platforms, one may set the following environmental variables in the `.*rc` file of choice - the example shown here is for the Bourne Again Shell (BASH) (see Section 4.3):

```
export ACCOUNT="fy424242"
export SBATCH_ACCOUNT=$ACCOUNT
export SALLOC_ACCOUNT=$ACCOUNT
export MOAB_ACCOUNT=$ACCOUNT
```

4.3 `.*rc` Configuration Files

In UNIX/Linux operating systems, there are shell configuration files which often start with a period and end with `rc`, which is short for "run commands." Due to the name starting with a period, one nickname for these files is "dotfiles." These files configure the initial settings for a user's account and can be very useful in specifying keyboard shortcuts and useful functions to improve productivity. The most commonly edited `.*rc` files are `.bashrc` for those who use the BASH shell (see Appendix D for an example) and `.cshrc` for those who use the C shell (CSH) (see Appendix E for an example).

However, there are certain pitfalls related to Sierra usage that should be avoided in `.*rc` files. First, NEVER load a `sierra` module (discussed more in Section 4.4) in a configuration file, as this can have undesirable effects on the operating environment, especially when other modules are loaded. Also, care must be taken when modifying the `PATH` variable, as `sierra` modules modify a user's `PATH` in order to point to code versions and Sierra tools. The best mode of operation is to keep the dotfiles as clean and minimal as possible so that the user environment does not conflict with the Sierra environment.

Other dotfiles exist for text editors (e.g., Vim and Emacs), Nvidia settings, mail utilities, etc., though these have little relevance in the context of Sierra usage.

4.4 Modules

Sandia platforms and the Sierra project make extensive use of modules, so it is useful to understand what modules are and how to use them. A module is a software tool that allows for dynamic modification of a user's environment. Once loaded, modules will alter or set shell environment variables such as `PATH` and `MANPATH`. Modules can also be easily *loaded* and *unloaded* dynamically for easy modification of the user environment. Thus, modules are attractive when running on different operating environments and for use in management of different application versions. The Sierra project uses modules for these reasons, both because Sierra is utilized on many different platforms and supports the use of different versions at any one time (see Chapter 6 for more information).

Following are some examples of common module commands as well as the output on different SNL systems. The list of available module often changes depending on the system being utilized. All available modules can be listed by using the following command:

```
$ module available
```

To see all Sierra-related modules, the following command may be used:

```
$ module available sierra
```

This command gives ALL `sierra` modules, including those used by developers. In order to list only the Sierra versions available for loading, add a trailing forward slash:

```
$ module available sierra/

---------------- /projects/sierra/linux_rh6/modules ----------------
sierra/4.30   sierra/4.30.1 sierra/4.32    sierra/4.32.1
sierra/4.33.1 sierra/4.33.2 sierra/master
```

Doing the following will load the `sierra` module, which will put the current default version of Sierra in your `PATH`:

```
$ module load sierra
```

Once modules have been loaded in a shell, the following command shows which modules are actually loaded:

```
$ module list
Currently Loaded Modulefiles:
  1) sntools/4.32                    6) sntools/master
  2) sierra-python/2.7               7) sierra-compiler/intel/12.1-2011.13.367
  3) sierra-git/1.7.3                8) sierra-mpi/intelmpi/4.1
  4) sierra-mkl/12.1-2011.13.367     9) sierra/master
  5) sierra-tbb/12.1-2011.13.367
```

The last set of useful module commands pertains to removing and switching modules. To remove a module from use, either of the following commands suffice:

```
$ module unload sierra
$ module purge
```

Please note that `module purge` is heavy-handed and will remove all loaded modules from the current shell session. This can have adverse effects on platforms where a number of environment modules are pre-loaded, so caution is advised.

To switch modules, use `switch` or swap[1]:

```
$ module switch sierra sierra/master
$ module swap sierra/master sierra/4.32
```

As a convenience, some of the module commands can be shortened. For example, `module available` can be shortened to `module avail` or even `module av` (at least two letters are necessary). This shortcut also applies to `list`, `swap`, and `switch`.

4.4.1 Just-in-Time Module Loading

There are times when a user needs to use different tools which require the use of different modules to set up the proper environment. However, in many circumstances it is dangerous to load multiple modules concurrently because some environmental setup conflicts may be introduced. One such example is the conflict caused when concurrently loading the Sierra and Dakota modules (see Section 5.10.2).

To avoid this issue, it is recommended to use a process known as Just-in-Time (JIT) module loading. This is best accomplished via scripting, and one simple setup is illustrated below:

1. Load the `dakota` module on a login script.

   ```
   $ module load dakota
   ```

2. Run the login script on a HPC login node which executes compute scripts that are submitted to the batch queue and run on compute nodes.

   ```
   $ ./login_script.sh
   ```

3. Within the compute scripts, unload the `dakota` module.

   ```
   $ module unload dakota
   ```

4. Load the `sierra` module within the compute script.

   ```
   $ module load sierra
   ```

[1]The `module swap` and `switch` commands have been shown to contain bugs on certain platforms. If encountered, please use the appropriate sequence of `unload` and `load`.

5. Run Sierra without queue submission using `launch` within the compute script.

```
$ launch -n 128 adagio -i input.i
```

6. Unload the `sierra` module within the compute script upon completion of the Sierra run.

```
$ module unload sierra
```

Since Dakota is often used as a wrapper application, it is recommended to use the Dakota User's Manual [3] as a reference. The Dakota manual contains an excellent chapter regarding parallelism use cases; the case described above is referred to as "Case 4" in the manual.

4.5 Stale Environments

Some users maintain the practice of keeping terminal windows with loaded modules open for a long time (on the order of weeks). While this practice is not necessarily bad, it can have negative consequences when running. If, for example, a module is altered by the module owner while a terminal window is open, the changes made will not be propagated to the terminal session, even though Sierra may expect the newer version of the module. This can result in difficult-to-decipher environmental error messages.

If you encounter odd error messages that seem to be related to the environment, the first step is to close the terminal and try the same set of commands in a new terminal window. If that does not change the outcome, then the next step is to log off the system and log back in to ascertain whether the error persists.

Chapter 5

Running Sierra

After the necessary accounts have been obtained and the user environment has been set up, Sierra may be utilized for simulations. From a Sierra-wide perspective, there are several considerations which are relevant to anyone who runs the Sierra suite of codes.

For information relating to specific application codes, please refer to relevant theory and user manuals, which can be found on the CompSim website (compsim.sandia.gov).

5.1 Commandline Invocation

Sierra applications (e.g., Sierra/SM, Sierra/Aero, Sierra/Thermal Multiphysics, Sierra/Low Mach Fluids, Sierra/SD) are standard Message Passing Interface (MPI) applications and can be invoked as such. For example, if your system uses OpenMPI (preferred by Sierra), one can launch a parallel application using `mpiexec`:

```
$ mpiexec --np 32 aria -i <input_file>.i
```

If your machine requires that MPI applications be submitted through a queueing system, you can invoke Sierra applications like any other MPI application on that system. For example, on a system using `sbatch` one way to launch the application is as follows:

```
$ /usr/bin/sbatch --nodes 32 --time 8:00:00 -A FY123456 mpiexec --np 32
   salinas <input file>.inp
```

5.2 `sierra` Script

While the above commands will work on many systems, they may need to be modified for each system and can be dramatically different depending upon the platform. Additionally, these commands only run the Sierra application of interest and will not decompose the mesh or concatenate the results. To simplify the use of Sierra on different systems, a script is available which wraps the pre-processing, run/submission, and post-processing steps into one command. The basic format of the command to invoke the script is as follows:

```
$ sierra <sierra_options> <APPLICATION> <application_options>
```

where `sierra_options` denotes commands that are passed to `sierra`, `APPLICATION` is the name of the application code, and `application_options` are the commands passed to the application code. Following is a list of common `sierra` options:

`--np`	Number of processors on which to run the simulation
`--make`	Build user-specified plugins and/or subroutines
`-a`	Preprocess the input file with aprepro (see Section 5.9.1)
`--queue-name`	Queue specification on HPC platforms
`-T`	Time limit for jobs run on HPC platforms
`--account`	WCID relevant to run
`--pre`	Execute mesh decomposition step
`--run`	Execute the simulation run step
`--post`	Execute the results concatenation step

By default, the `sierra` script will execute all three of the last options listed above, so it is only necessary to include one or two of the options when all three steps are not required. Also note that this is not all of the `sierra`-specific options, and that a complete list of the available options on a certain platform can be found by issuing the command

```
$ sierra --help
```

The most commonly-used `application_options` option is `-i`, which specifies the input file that is used by the application. To find all available options for a specific application, the following should be executed:

```
$ <APPLICATION> --help
```

Following are some common examples of `sierra` script usage:

```
$ sierra --np 8 fuego -i input.i
$ sierra --run aria -i input.i
$ sierra --np 128 -T 24:00:00 --queue-name nw adagio -i input.i
```

Please note that the `sierra` script is a convenience script and may not function well in complex situations, such as non-standard run specifications or code couplings. In situations of non-standard run specifications, the best place to look for examples is on HPC platforms at the following location:

```
/home/samples/<platform_name>/
```

For code couplings that cannot be accomplished through `Solution Control`, like Dakota-Sierra or Aero-SD, examples can be supplied by contacting Sierra Help. See Section 5.10 for more information on how to couple Sierra with other codes.

5.3 Pre- and Post-processing

One aspect of running simulations in parallel is splitting the model into pieces for each processor. For standard Sierra simulations, the SEACAS `decomp` utility (see Section 5.9.1) will split the input mesh(es) into the same number of parts as the number of processors to run on. The following command will accomplish this as a separate step to running the simulation:

```
$ decomp --processors <NP> <mesh_name>.ext
```

There has also been effort into making mesh decomposition a part of each of the Sierra application codes. This is also referred to as "on-the-fly" mesh decomposition and eliminates the need to have a distinct set of decomposed mesh files. In order to activate this capability, the following needs to be added to each `Finite Element Model` specification in the input file:

```
Decomposition Method = <method>
```

There are several options for `method`, but the most common are `RIB` and `RCB`. If the `sierra` script is being used, make sure to specify `--run` so that the standard decomposition step is avoided.

In a similar fashion, the post-processing step of Sierra simulations involves concatenating the decomposed results files into one file. This is accomplished using the SEACAS utility `epu` (see Section 5.9.1). The simplest command for using `epu` is shown below - this example illustrates combining Exodus files with the naming convention `output.e.10.[00-15]` from a sixteen-processor Sierra run:

```
$ epu -auto output.e.16.00
```

The "on-the-fly" method also works for results concatenation and requires additional lines in the input file, in each of the `Results Output` blocks:

```
Property COMPOSE_RESULTS = Yes
```

The following lines are also required in `Restart Data` blocks:

```
Property COMPOSE_RESTART = Yes
Decomposition Method = <method>
```

For the `Restart Data` block(s), ensure that `method` corresponds to the `method` in the `Finite Element Model` block.

"On-the-fly" decomposition and concatenation will at some point become the default behavior in Sierra, though there is no hard date for when this will happen.

5.3.1 Signals

All Sierra applications have the capability of exiting gracefully upon reception of certain POSIX signals, e.g., SIGKILL. The line which enables this for each type of output block is as follows:

```
Output on Signal = <SIGNAL>
```

Signal 10 (SIGUSR1) is the preferred signal to instruct Sierra applications to exit gracefully, as this signal is not commonly used by the system.

Note that signals do no always get passed to applications by the resource manager on HPC platforms so the use of signals may be problematic.

5.4 Restart

One important consideration when running on any system, and especially HPC platforms, is maximization of throughput. For example, if hardware fails or the application walltime is reached, a user would like to restart their simulation at or near the time when the run was terminated. Sierra provides a checkpoint/restart capability to provide for such scenarios, and it behooves all analysts to implement this capability in their models.

The standard and well-tested method of utilizing restart is to write a restart file periodically during the run, which is of exodus format and contains all fields necessary to start at a point midway in the simulation. These restart files are typically decomposed into the number of processors on which the simulation is performed. Below is a typical restart block for many Sierra applications which should be placed in the Region scope of the input file:

```
Begin Restart Data restartData
    Input Database Name  = model.rso
    Output Database Name = model.rso
    Start Time           = 0.0    # seconds
    At Step 0 Increment  = 1000.0 # steps
    Additional Times      = 1250.0 # seconds
End   Restart Data restartData
```

Restart files should not be written too often, as they only serve to reduce wasted computation time in the event of failure. Before running a model, consider the problem size and the amount of time you are willing to lose in the event of failure - for a long-running problem (e.g., 96 hours), that number may be 6 wall-clock hours, while for a shorter time limit (e.g., 24 hours) the time may be closer to 2 wall-clock hours. Once you have decided the frequency of restart output, you then will need to determine the approximate number of steps that occur in that wall time and provide that frequency in the Restart block. There has been discussion about providing a wall-clock frequency in Sierra input files, but this has not yet been implemented.

In order to use a restart file for its intended purpose of running a model later in simulation time, one of the following two lines must be added to the Sierra scope of the input file:

```
Restart = Auto
```

or

```
Restart Time = <time>
```

The behavior of each method will be explained in the following sections.

5.4.1 Automatic Restart

You can use the restart option in an automated fashion by using a combination of the `Restart` command line in the Sierra scope and the `Database Name` command line in the `RESTART DATA` command block. This automated use of restart can best be explained by an example. We will use a two-processor example and assume all files will be in our current directory.

The option of automated restart will not only manage the restart files to prevent overwriting, it will also manage the results files and history files to prevent overwriting. In the example we give, we will assume our run includes a `Results Output` command block with the command line

```
Database Name = rslt.e
```

to generate results files with the root file name `rslt.e`. We will also assume a run includes a `History Output` command block with the command line

```
Database Name = hist.h
```

to generate history files with the root file name `hist.h`.

For the first run in our restart sequence, we will have the command line

```
Restart = Automatic
```

in the Sierra scope of our input file. In a time stepping command block, embedded in a `Solution Control` command block in the Procedure scope of our input file, we will have the command line:

```
Start Time = 0.0
```

In the `Time Control` command block we will have the command line

```
TERMINATION TIME = 2.5E-3
```

to set the limits for the begin and end times of the first restart run. These time-related command lines should not be confused with the `Start Time` and `Termination Time` command lines that appear in the `Restart Data` command block.

Finally, for the first run in our restart sequence, the `Restart Data` command block in our input file will be as follows:

```
Begin Restart Data Restart_data
  Database Name = g.rsout
  At Time 0.0 Increment = 0.25E-3
End Restart Data Restart_data
```

In this block, the `Database Name` command line specifies a root file name for the restart file. The `At Time` command line gives the time when we will start to write the restart information and the interval at which the restart information will be written. All the same commands used to define when results output is written can also be used in restart output.

For our first run, the automatic restart option will generate the restart files:

```
# restart files
g.rsout.2.0
g.rsout.2.1
# results files
rslt.e.2.0
rslt.e.2.1
# history files
hist.h.2.0
hist.h.2.1
```

For the above files, there are extensions on the file names that indicate we have a two-processor run. The 2.0 and 2.1 extensions associate the restart files with the corresponding mesh files on each processor. (If our mesh file is `mesh.g`, then our mesh files on the individual processors will be `mesh.g.2.0` and `mesh.g.2.1`.) All restart information in the above files appears at time intervals of 0.25×10^{-3}, and the last restart information is written at time 2.5×10^{-3}. We have also listed the results and history files that will be generated for this run due to the file definitions in the command blocks for the results and history files.

For the second run in our sequence of restart runs, we want to start at the previous termination time, 2.5×10^{-3}, and terminate at time 5.0×10^{-3}. We leave everything in our input file (including the `START TIME = 0.0` command line in the `TIME STEPPING` command block, the `RESTART` command line, and the `RESTART DATA` command block) the same except for the `TERMINATION TIME` command line (in the `TIME CONTROL` command block). The `TERMINATION TIME` command line will now become:

```
Termination Time = 5.0E-3
```

The actual start time for the second run in our analysis is now set by the last time (2.5×10^{-3}) that restart information was written. The command line `Start Time = 0.0` in the `Time Stepping` command block is now superseded as the actual starting time for the second run by the restart commands. Any `Start Time` command line in a `Time Stepping` command block is still valid in terms of defining time stepping blocks (these blocks being used to set activation periods), but the restart process sets the actual start time for our analysis. This pattern of control for setting the actual start time holds for any run in our sequence of restart runs.

For the second run in our sequence of restart runs, the restart files will be from time 2.5×10^{-3} to time 5.0×10^{-3}. The restart files in our current directory after the second run will be as follows:

```
# restart files
g.rsout.2.0
g.rsout.2.1
g.rsout-s0002.2.0
g.rsout-s0002.2.1
# results files
rslt.e.2.0
rslt.e.2.1
rslt.e-s0002.2.0
rslt.e-s0002.2.1
# history files
hist.h.2.0
hist.h.2.1
hist.h-s0002.2.0
hist.h-s0002.2.1
```

Notice that we have generated new restart files with a `-s0002` extension in addition to the extension associated with the individual processors. All restart information in the above files with the `-s0002` extension appears at time intervals of 0.25×10^{-3}, the restart information is written between time 2.5×10^{-3} and time 5.0×10^{-3}, and the final restart information is written at time 5.0×10^{-3}. The restart files for the first run in our sequence of restart runs, `g.rsout.2.0` and `g.rsout.2.1`, have been preserved. New results and history files have been created using the same extension, `-s0002`, as that used for the restart files. The original results and history files have been preserved.

Now, we want to do a third run in our sequence of restart runs. For the third run in our sequence of restart runs, we want to start at the previous termination time, 5.0×10^{-3}, and terminate at time 8.5×10^{-3}. We leave everything in our input file (including the `Start Time` command line, the `Restart` command line, and the `Restart Data` command block) the same except for the `Termination Time` command line. The `Termination Time` command line (within the `Time Control` command block) will now become:

```
Termination Time = 8.5E-3
```

For the third run in our sequence of restart runs, the restart files will be from time 5.0×10^{-3} to time 8.5×10^{-3}. The restart files in our current directory after the third run will be as follows:

```
# restart files
g.rsout.2.0
g.rsout.2.1
g.rsout-s0002.2.0
g.rsout-s0002.2.1
g.rsout-s0003.2.0
g.rsout-s0003.2.1
# results files
rslt.e.2.0
rslt.e.2.1
rslt.e-s0002.2.0
rslt.e-s0002.2.1
rslt.e-s0003.2.0
rslt.e-s0003.2.1
# history files
hist.h.2.0
hist.h.2.1
hist.h-s0002.2.0
hist.h-s0002.2.1
hist.h-s0003.2.0
hist.h-s0003.2.1
```

Notice that we have generated new restart files with a `-s0003` extension in addition to the extension associated with the individual processors. All restart information in the above files with the `-s0003` extension appears at time intervals of 0.25×10^{-3}, the restart information is written between time 5.0×10^{-3} and time 8.5×10^{-3}, and the final restart information is written at time 8.5×10^{-3}. The restart files for the first and second runs in our sequence of restart runs have been preserved. New results and history files have been created using the same extension, -s0003, as that used for the restart files. The original results and history files have been preserved.

The process described can be continued as long as necessary. We will continue the process of generating new restart files with extensions that indicate their place in the sequence of runs.

5.4.2 Time-specified Restart

You can use the restart option and select specific restart times and specific restart files to read from and write to by using a combination of the Restart Time command line in the Sierra scope and the Input Database Name and Output Database Name command line in the Restart Data command block. This "controlled" use of restart can best be explained by an example. We will use a two-processor example and assume all files will be in our current directory. In this example, we will manage the creation of new restart files so as not to overwrite existing restart files. Unlike the automated option for restart, this controlled use of restart requires that the user manage restart file names so as to prevent overwriting previously generated restart files. The same is true for the results and history files. The user will have to manage the creation of new results and history files so as not to overwrite existing results and history files. Creating new results and history files for each run in the sequence of restart runs requires changing the Database Name command line in the Results Output and History Output command blocks. We will not show examples for use of the Database Name command line in the Results Output and History Output command blocks here, as the actual use of the Database Name command line in the results and history command blocks would closely parallel the pattern we see for management of the restart file names.

For the first run in our restart sequence, we will have only a Restart Data command block in the region; there will be no restart-related command line in the Sierra scope of our input file. We will, however, have a

```
Start Time = 0.0
```

command line in a Time Stepping command block (within the Time Control command block) and a

```
Termination Time = 2.5E-3
```

command line within the Time Control command block to set the limits for the begin and end times. The Restart Data command block in our input file will be as follows:

```
Begin restart Data Restart_data
  Output Database Name = RS1.rsout
  At Time 0.0 Increment = 0.5e-3
End Restart Data Restart_data
```

For our first run, the restart option will generate the restart files:

```
RS1.rsout.2.0
RS1.rsout.2.1
```

For the above files, the extensions on the file names indicate that we have a two-processor run. The 2.0 and 2.1 extensions associate the restart files with the corresponding mesh files on each processor. If our mesh file is `mesh.g`, then our mesh files on the individual processors will be `mesh.g.2.0` and `mesh.g.2.1`. All restart information in the above files appears at time intervals of 0.5×10^{-3}, and the last restart information is written at time 2.5×10^{-3}.

For the second run in our sequence of restart runs, we want to start at the previous termination time, 2.5×10^{-3}, and terminate at time 5.0×10^{-3}. To do this, we must add a

```
Restart Time = 2.5E-3
```

command line to the Sierra scope and set the termination time to 5.0×10^{-3} by using the command line

```
Termination Time = 5.0E-3 \rm
```

within the `Time Control` command block.

The actual start time for the second run in our analysis is now set by the restart time set on the `Restart Time` command line, 2.5×10^{-3}. The command line `Start Time = 0.0` in the `Time Stepping` command block is now superseded as the actual starting time for the second run by the restart commands. Any `Start Time` command line in a `Time Stepping` command block is still valid in terms of defining time stepping blocks (these blocks being used to set activation periods), but the restart process sets the actual start time for our analysis. This pattern of control for setting the actual start time holds for any run in our sequence of restart runs.

We also must change the `Restart Data` command block to:

```
Begin Restart Data Restart_data
  Input Database Name = RS1.rsout
  Output Database Name = RS2.rsout
  At Time 0.0 Increment = 0.5E-3
End Restart Data Restart_data
```

For this second run, we will read from the files:

```
RS1.rsout.2.0
RS1.rsout.2.1
```

And we will write to these files:

```
RS2.rsout.2.0
RS2.rsout.2.1
```

All restart information in the above output files, `RS2.rsout.2.0` and `RS2.rsout.2.1`, appears at time intervals of 0.5×10^{-3}, restart information is written from time 2.5×10^{-3} to time 5.0×10^{-3}, and the last restart information is written at time 5.0×10^{-3}. Notice that we have preserved the restart files from the first run from our restart sequence of runs because we have specifically given the input and output databases distinct names — `RS1.rsout` for the input file name and `RS2.rsout` for the output file name.

Now, we want to do a third run in our sequence of restart runs. For this third run, we want to start at time 4.5×10^{-3} and terminate at time 8.5×10^{-3}. We do not want to start at the termination time for the previous restart, which is 5.0×10^{-3}; rather, we want to start at time 4.5×10^{-3}. We change the `Restart Time` command line to

```
Restart Time = 4.5E-3
```

and the `Termination Time` command line within the `Time Control` command block to:

```
TERMINATION TIME = 8.5E-3
```

And we change the `Restart Data` command block to:

```
Begin Restart Data Restart_data
  Input Database Name = RS2.rsout
  Output Database Name = RS3.rsout
  At Time 0.0, Increment = 0.5E-3
End Restart Data Restart_data
```

For this third run, we will read from these files:

```
RS2.rsout.2.0
RS2.rsout.2.1
```

And we will write to these files:

```
RS3.rsout.2.0
RS3.rsout.2.1
```

All restart information in the above output files, `RS3.rsout.2.0` and `RS3.rsout.2.1`, appears at time intervals of 0.5×10^{-3}, restart information is written from time 4.5×10^{-3} to time 8.5×10^{-3}, and the last restart information is written at time 8.5×10^{-3}. Notice that we have preserved all restart files from previous runs in our restart sequence of runs because we have specifically given the input and output databases distinct names for this third run.

5.5 User Subroutines

Many of the Sierra code executables may be extended for the users' needs via user subroutines, which may be written in either Fortran, C, or C++, depending upon the executable. For details on how to write user subroutines for a Sierra application of interest, please refer to the respective User Manual.

To compile a Sierra user subroutine, the `sierra` script may be invoked with the `--make` option:

```
$ sierra --make aria -i input.i
```

The subroutines which are built are dependent upon the input file, which will contain lines like this:

```
User Subroutine File = userSub.F
Load User Plugin File userSub.C
```

The first line will create a completely new binary which incorporates `userSub.F` and will place this binary within `./UserSubsProject/bin`. This new binary executable may then be invoked with the following command:

```
$ sierra ./UserSubsProject/bin/adagio -i input.i
```

The second input file line will take the user subroutine `userSub.C` and create a shared object file `userSub.so` which will be dynamically linked to the executable that was used to compile the subroutine. The model will then be run as normal.

It is important to note that the subroutine must not be used with a different executable than the one which was used to compile it. For example, a model with a user subroutine that was compiled with version 4.33.2 must be run only with version 4.33.2. If one runs their model with a different version of the code than that which was used to compile a user subroutine, at best the code will terminate, and at worst, the code will run but may give erroneous results. The safest course of action is to re-compile any user subroutines each time the code is run in order to ensure agreement between the compiled subroutine and the code.

Some platforms, such as the Advanced Technology Systems (ATS) platforms, may have different requirements for running Sierra with user subroutines. In some instances, running with user subroutines may not be possible, so care is advised when planning activities on different or new platforms.

5.6 Memory Considerations

A significant factor that must be taken into account when running any simulation code, including Sierra, is the memory use of the application with respect to the available memory on the platform being used. For instance, one would like to avoid situations of memory overrun, which will terminate the run and may corrupt the platform. On the other hand, it is important to utilize memory as efficiently as possible, as analyst throughput has been demonstrated to improve for many applications on HPC platforms [4].

All Sierra applications provide memory diagnostics in the log file, though the output may differ slightly. An example of the memory output in a Sierra application log file during run-time, which is output periodically throughout the simulation, is shown below:

```
Memory Usage: current = 267747328 (255.3 M), high-water-mark = 267747328 (255.3 M)
```

At the successful end of a simulation, a more complete memory summary is output to the log file and looks like the following:

```
Memory summary of 4 processors
           Metric              Largest     Processor     Smallest     Processor
---------------------------- --------- ---------------- --------- ----------------
Dynamically Allocated (Heap)  106.3 MB on 1 at 1:34.652  105.1 MB on 0 at 1:34.652
Largest Free Fragment (Heap)   48.2 MB on 1 at 1:34.652   47.6 MB on 0 at 1:34.652
         Total Memory In Use  518.7 MB on 0              518.3 MB on 1
           Major Page Faults    6 flts on 3                6 flts on 1


Performance metric summary
---------------------------------------------------
Min High-water memory usage 256.6 MB
Avg High-water memory usage 257.2 MB
Max High-water memory usage 257.9 MB

Min Available memory per processor 4036.1 MB
Avg Available memory per processor 4036.1 MB
Max Available memory per processor 4036.1 MB

Min No-output time 27931.2 sec
Avg No-output time 27935.9 sec
Max No-output time 27937.7 sec
```

```
--------------------------------------------------
```

The first table shows information about maximum and minimum memory usage plus the total amount of memory used on the processors, including Sierra as well as system processes. In this example, the total memory in use on four processors varied from 518.3 to 518.7 MB. The variation of Sierra memory usage is shown under the `Performance metric summary`, where it can be seen that the Sierra application used between 256.6 and 257.9 MB. In cases where the load is poorly balanced between processors, the variation in memory usage may increase and should be investigated further.

Because Sierra is run on a wide variety of systems with differing memory configurations, the available memory per processor is also output. On the system where this example was run, the maximum theoretical available memory is 4036.1 MB, or roughly 4.0 GB. The actual memory available will always be lower than this due to system processes, but this mark is useful for showing an estimation of how many processors are required to maximize memory efficiency.

5.7 I/O

During Sierra application runs, some form of input/output (I/O) must be executed. Common forms of I/O from Sierra applications include mesh read, mesh transfers, results output, heartbeat output, history output, and restart. Depending on the scale of the analysis, each of these types of I/O have an associated computational cost, and so must be considered with care.

5.7.1 Good Citizenship

I/O can be one of the most computationally intensive operations in a Sierra run, so it must be used appropriately. For instance, writing a restart file every iteration can quickly bog down a system and a simulation, especially when a large model is being run on HPC platforms with specialized filesystems. This adversely affects other users and in some instances has even caused HPC platforms to slow to the point of usability.

To maximize efficiency in I/O, it is best to determine the least amount of required I/O, which is specific to each model analysis. Consider the following example:

> A large simulation will be run for several days on an HPC system. Data probes will provide important information at key points in the model and must have a high amount of temporal information. Some key animations and graphics will also be made for certain simulation times. In order to prevent the loss of data in the case of a system failure or model abort, restart will be required. This simulation will require the following output at the listed frequencies:

- Every iteration – heartbeat output (data probes)
- Every 100 time steps – results output (animations/graphics)
- Every 2 hours of wall-clock time – restart output

It is also recommended to stagger restart and results output, as concurrent output of these types can be very taxing on the filesystem.

5.7.2 Striping

Striping is a distributed filesystem term for how many pieces a file will be broken up into and placed within different object storage targets (OSTs). The default striping on several HPC platforms is 4. The example shown below shows how to modify striping and determine what striping is already present. If you have any files that are larger than about 1.5TB, you will want to make sure your striping is set to a value larger than 1. Currently on TLCC2 Chama, the OSTs have, on average, 1.8TB of free space. So, with striping set to 1, each file will need to be smaller than what is available to fit onto an OST or problems will arise.

```
$ hostname
chama-login4
$ pwd
/gscratch2/rpshaw
$ mkdir test_1 test_2
$ lfs getstripe test_*
test_1
stripe_count:   4 stripe_size:    1048576 stripe_offset:  -1
test_2
stripe_count:   4 stripe_size:    1048576 stripe_offset:  -1
$
$ ### USE setstripe TO SET STRIPING ###
$
$ lfs setstripe -c 1 test_1
$ lfs getstripe test_*
test_1
stripe_count:   1 stripe_size:    1048576 stripe_offset:  -1
test_2
stripe_count:   4 stripe_size:    1048576 stripe_offset:  -1
$ cd test_1
$ echo "TESTING" > test_1.txt
$ lfs getstripe test_1.txt
test_1.txt
lmm_stripe_count:   1
lmm_stripe_size:    1048576
lmm_stripe_offset:  51
```

```
        obdidx           objid          objid            group
            51         1435257       0x15e679                0

$ cd ../test_2
$ echo "TESTING" > test_2.txt
$ lfs getstripe test_2.txt
test_2.txt
lmm_stripe_count:   4
lmm_stripe_size:    1048576
lmm_stripe_offset:  16
        obdidx           objid          objid            group
            16         1448312       0x161978                0
            74         1393623       0x1543d7                0
            88         1436155       0x15e9fb                0
           132         1448615       0x161aa7                0
```

A number of HPC system administrators and Sierra I/O experts have noticed faster I/O with striping set to 1 on distributed filesystems for most I/O situations. If you perform a considerable amount of filesystem I/O and all individual files are smaller than the OST size (roughly 1.5 TB), then it would make most sense to set the file striping to 1.

Note that setting the file striping on an existing directory will not change the striping on files or directories previously created inside that directory. Thus, the best idea is to create a new directory and change its file striping, then perform analyses within the newly-created directory.

5.8 Interpreting Terminal Output

In addition to the user-specified output outlined in Section 5.7, automatically generated output is given to aid users in charting simulation progress and debugging model issues. These tools are very valuable if you know how to interpret and utilize them.

In all cases where a job has died and you cannot determine the cause, please send **all** of types of automatically generated files to Sierra Help in order to expedite the investigation. Note, however, that Sierra Help email is limited to files less than 64 MB in size. For larger files, please send the file location and change the file permissions, if necessary (see Section 3.4), to allow the files to be accessed. In the case of classified or need-to-know (NTK) permission issues, note that in your support request.

5.8.1 Log File

All Sierra applications provide varying forms of log file output of overall simulation information, solver status, variable output, and other data. The preamble of each Sierra log file is essentially the same and gives useful information such as code version, run start time, and input file parsing. An example of this preamble is shown below.

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----

                                      Aria

                 Coupled multiphysics including Navier-Stokes,
                elasticity, energy transport, species transport,
                  electrostatics;  free and moving boundaries;
                           transient or steady state.

                          Version 4.33.1-398-gaeb9bfbb

                       With coupled mechanics support for
                          Aria - Coupled multiphysics
                 Encore - Solution Verification Analysis Region

                          Sandia National Laboratories
               Albuquerque, New Mexico and Livermore, California

                      Please email questions and comments to
                            sierra-help@sandia.gov


                 Notice: This computer software was prepared by
                Sandia Corporation, hereinafter the Contractor,
                   under Contract DE-AC04-94AL85000 with the
                 Department of Energy (DOE).  All rights in the
                computer software are reserved by DOE on behalf
                    of the United States Government and the
                Contractor as provided in the Contract. You are
                  authorized to use this computer software for
                   Governmental purposes but it is not to be
                 released or distributed to the public. NEITHER
                 THE U.S.  GOVERNMENT NOR THE CONTRACTOR MAKES
                ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY
                    LIABILITY FOR THE USE OF THIS SOFTWARE.

+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----

```

```
    Directory /home/rpshaw/foamDecomposition/
   Executable /projects/sierra/linux_rh6/install/master/bin/aria
        Built May 12 2014 18:46:01
Build Options linux intel-12.1 release
  Run Started May 14 2014 11:04:22
         User rpshaw
 Architecture cee-build001
         Host cee-build001
     Hardware x86_64
      Running Linux
   Processors 1
   Processing foamDecomposition.i

    Product                Version           Qualifier
---------------- ------------------------- ----------
            ACME 2.9.0
            Aria 4.33.1-398-gaeb9bfbb
       Chaparral 3.3.1 development         unreleased
          Encore 4.33.1-398-gaeb9bfbb
             FEI 2.24.02
            GDSW Dev
           Linux 2.6.32-358.2.1.el6.x86_64
            MPIH 4.33.1-398-gaeb9bfbb
SIERRA Framework 4.33.1-398-gaeb9bfbb
        Trilinos 11.9.0
Trilinos|AztecOO 11.9.0
      UtilityLib 4.33.1-398-gaeb9bfbb
          Zoltan 3.8
```

Here it can be seen that Aria version `4.33.1-398-gaeb9bfbb` is used, and further, it can be seen that the `master` (i.e., VOTD) version is used based on the executable path:

```
/projects/sierra/linux_rh6/install/master/bin/aria
```

It can also be seen that the run was started at `May 14 2014 11:04:22` on `cee-build001` with 1 processor in `/home/rpshaw/foamDecomposition/`.

For more information on interpreting application-specific log file output, please refer to the application user manuals.

5.8.2 STDOUT and `slurm-*` Files

In addition to the standard log file output, several messages may appear to standard output (STDOUT) or to a job file if on a queued system. For Sandia queued systems, this is typically labeled `slurm-JOBID.out` due to Sandia's use of SLURM as a job scheduler. One of the most common error messages in this output is something like the following, where the signal number, processor number and date may be changed:

```
Sierra received signal 11 at exception on processor 52:
Segmentation violation error

Sierra received signal 11 at Wed Apr 17 15:53:42 2013
Wed Apr 17 15:53:42 2013
```

While in and of itself this error message is not overly helpful in diagnosing the root problem of the run termination, it does show that the problem did terminate with a segmentation fault and was killed with POSIX signal 11 (SIGSEGV), which also denotes an invalid memory reference. For a list of all signals and their numeric mapping, execute the following command:

```
$ kill -l
```

Another message that may be useful on systems where jobs have a time limit looks like this:

```
slurmd[rs1779]: *** JOB 12424308 CANCELLED AT 2013-03-08T07:21:06 DUE TO
   TIME LIMIT ***
```

If the job appears to have suddenly halted without any apparent error in the log file, a quick check for this message may help to give the reason for job termination.

5.9 Other Tools

Other tools are available which assist in different steps of running Sierra applications. This section is not meant to be an exhaustive list of all available tools and users should refer to documentation specific to the tools mentioned for more detail.

5.9.1 SEACAS

SEACAS [5] is a set of pre- and post-processing codes that have been developed in tandem with Sierra. From the main SEACAS page:

> "The Sandia National Laboratories (SNL) Engineering Analysis Code Access System (SEACAS) is a collection of structural and thermal codes and utilities used by analysts at SNL. The system includes pre- and post-processing codes, analysis codes, database translation codes, support libraries, UNIX shell scripts, and an installation system."

SEACAS codes that are commonly used by Sierra analysts include the following:

Aprepro Aprepro is an algebraic pre-processor that reads a file containing both general text and algebraic, string, or conditional expressions. It interprets the expressions and outputs them to the output file along with the general text. Aprepro contains several mathematical functions, string functions, and flow control constructs. In addition, functions are included that, with some additional files, implement a units conversion system and a material database look-up system.

Decomp Decomp decomposes an Exodus-formatted mesh file into partitioned mesh files for use in parallel computation. Decomp includes several different decomposition methods for different application needs.

EPU EPU combines multiple Exodus databases produced by a parallel application into a single Exodus database.

Exotxt Exotxt converts Exodus-formatted binary files to text files; a similar program exists called Txtexo which performs the opposite transformation.

Grope Grope is a program that examines the input to a finite element analysis (which is in the Genesis database format) or the output from an analysis (in the Exodus database format). Grope allows the user to examine any value in the database. The display can be directed to the user's terminal or to a print file.

For more information on SEACAS, see the documentation at the following URL:

`http://compsim.sandia.gov/compsim/Docs/SEACAS/index.html`

5.9.2 Percept

Percept is a toolkit for verification that includes a standalone code (`mesh_adapt`) that can refine very large meshes in parallel. The tool can be run from the command line and is completely automated. Features include support for a variety of element types, use of CAD geometry files, maintenance of mesh size gradings, smoothing of meshes, and output of mesh size statistics. More information regarding Percept and some of the available tools can be found on CompSim.

5.9.3 Catalyst

Catalyst [6] is a visualization co-processing library developed by Kitware that is essentially a lightweight ParaView [1] server library. Efforts have been made at Sandia to integrate Catalyst tightly with many Sierra application codes, including using the same input syntax as one would normally find in a Sierra Results Output block. For more information regarding this capability, please contact catalyst-help@sandia.gov.

5.9.4 CUBIT

CUBIT is "a full-featured software toolkit for robust generation of two- and three-dimensional finite element meshes (grids) and geometry preparation. Its main goal is to reduce the time to generate meshes, particularly large hex meshes of complicated, interlocking assemblies. It is a solid-modeler-based pre-processor that meshes volumes and surfaces for finite element analysis." [7]

Many analysts use this Sandia-developed mesh generation tool to create and modify computational meshes for use in Sierra. CUBIT also employs Aprepro to perform algebraic operations, which lends to its use with Sierra applications. CUBIT is available at Sandia and can be obtained for government use for free. For help with CUBIT, visit their web page or contact Cubit Help at cubit-help@sandia.gov.

5.9.5 Sandia Analysis Workbench (SAW)

The Sandia Analysis Workbench (SAW) was created to help analysts build a computational model using Sierra. The SAW "environment is a collection of integrated software tools to help analysts reduce the time required to build, analyze, and understand complex finite element analysis simulations. Depending upon the specific edition, it provides convenient graphical model building and meshing capabilities, resource links, file management, metadata editor, tools for teaming, the ability to add engineering notes to objects, and data management capabilities." [8].

SAW is available to all Sandia users of Sierra and can also be included in Sierra distributions for external users of the Sierra codes. For more information on SAW, please contact dart-help@sandia.gov.

5.9.6 More Information

More information regarding non-Sierra codes that can be used with Sierra applications can be found at this URL for internal Sandia analysts:

`https://computing.sandia.gov/codes_and_tools`

5.10 Coupling within Sierra and with Non-Sierra Codes

One of Sierra's aims is to be a multi-physics suite of codes, and it has the capability to couple some of its own applications and to be coupled with certain non-Sierra codes when the need arises. Sierra can also be utilized for the purposes of verification and parameter studies, which requires integration with optimization packages. This section outlines some of these couplings and gives recommendations for achieving the desired coupling.

5.10.1 Multi-physics Coupling

To enable multi-physics simulations, several Sierra applications may be coupled to each other through various means. One means of doing this is via framework `Solution Control` - examples of this include Sierra/Thermal Multiphysics and Sierra/SM (`arpeggio`) and Sierra/Thermal Multiphysics and Sierra/Low Mach Fluids (`fuego_aria`). The coupling of Sierra/SD and Sierra/Aero is accomplished not via a combined executable, but through a Multiple Program Multiple Data (MPMD) model. These couplings of Sierra-Sierra applications are described in the appropriate user manuals.

Coupling of CTH [9] to some Sierra applications is also available in the ITAR distribution of Sierra. Sierra/SM couples to CTH via the `fortissimo` executable. Aero also provides one-way coupling to CTH via the reading in of specific files. The specifics of these couplings can also be found in the Sierra/SM and Sierra/Aero manuals.

Other non-Sierra codes have also been coupled to Sierra. Contact Sierra Help if interested in determining whether a specific non-Sierra coupled application exists or would be feasible to implement.

5.10.2 Dakota: Optimization & Solution Verification

Sierra analysts who desire to perform sensitivity analysis, design optimization, or uncertainty quantification may use Dakota [10]. Setting up Dakota with Sierra requires an understanding of the desired workflow and computing environment. See Section 4.4.1 for information regarding running Sierra and Dakota together.

Chapter 6

Versioning

The Sierra codes follow a strict versioning system in order to meet software quality assurance (SQA) demands. An illustration of Sierra's release process may be seen in Figure 6.1. The general numbering notation that Sierra follows is X.YY.Z, where X denotes the major number, YY denotes the minor number, and Z denotes the point number. These numbers may have different meanings depending upon the type of version and are discussed in the following sections.

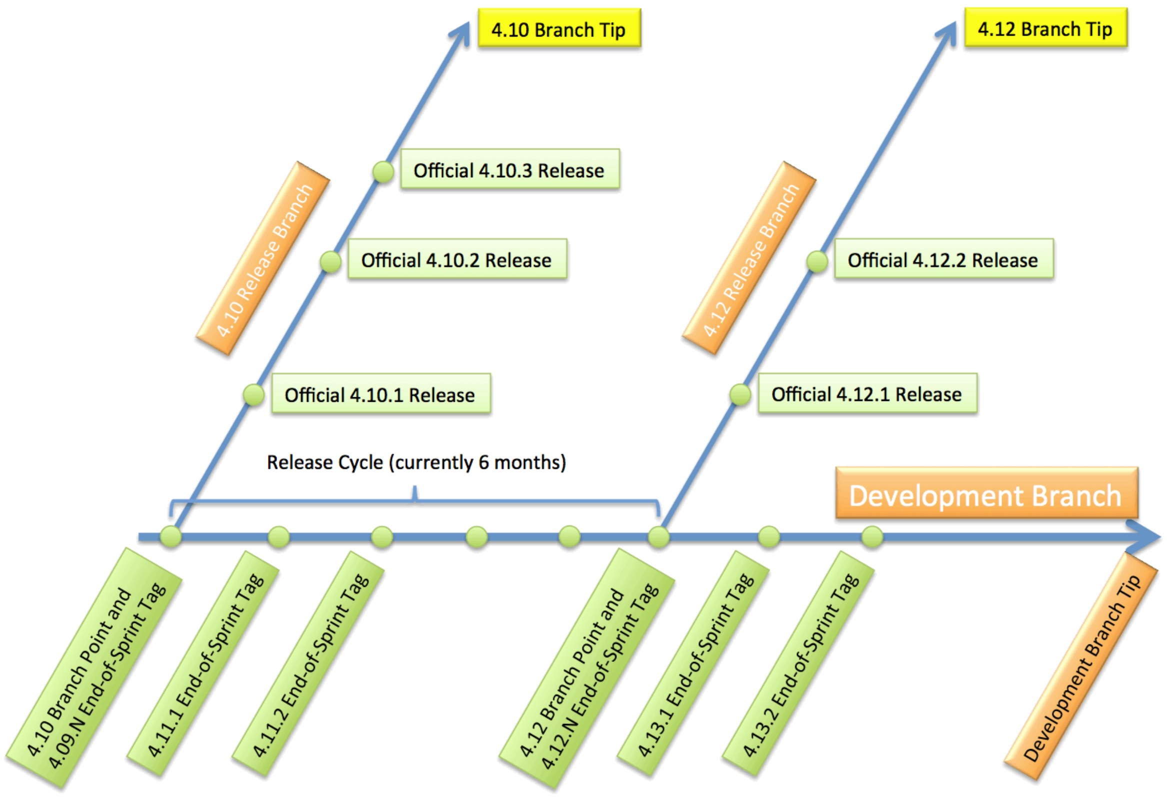


Figure 6.1. Illustration of Sierra release process.

6.1 Release

Sierra follows a release cycle that is roughly six months in duration. At the end of each release cycle, the Sierra developers create a new release branch, which has been rigorously tested to remove as many issues as possible. Sierra releases follow the X.YY notation, where YY is an even number, e.g., 4.10. After the release branch has been created, further testing is performed, and when consensus is reached between all Sierra development team leads, then a (static) point release is made and follows the numbering convention X.YY.Z, where Z denotes the point number and begins with 1. For example, the first point release of the 4.10 branch is 4.10.1.

Note that in this example, 4.10 (without the trailing point number) is a *development branch*, i.e., changes can be made to it. If serious code bugs are found in the 4.10.1 point release, a code fix can be developed and applied as a patch to the 4.10 release branch. Another point release can be created that includes the fix, which would be labeled 4.10.2.

For production Sierra analysis and for most applications, the release version of Sierra is the recommended for all code use. However, there are situations where using a newer, less stable version of Sierra is warranted:

- New code features have been added since the release, and a user would like to test the feature on their problem.
- A bug fix was made in a newer version which allows a model to run.

It is important to note that versions of the code newer than the point release have not been as rigorously tested and may provide incorrect results.

As a policy, Sierra supports the two most recent releases at any time. Some users may need to use an older release for qualification purposes, so this policy should be taken into account when planning these activities. If special support is needed for longer than the official policy, please let Sierra Help know so that your needs can be correctly communicated.

Please note that backward compatibility of user input is not always guaranteed; users should use release notes on CompSim as guidance on such matters. Also, due to frequent code changes (e.g., bug fixes, new features), results may differ between different releases - this can occur if the suite of tests do not test similar user input setup or are not run at the scale of the model of concern. If the results are different enough or are trending away from expected answers, please contact Sierra Help so that they may raise the issue with the development teams.

6.1.1 Default Version

The default version of Sierra corresponds to the code version loaded into a user's working environment when the `sierra` module is loaded:

```
$ module load sierra
```

The Sierra development operations (DevOps) team, which manages Sierra builds and distributions, also manages the default version that will be loaded with the `sierra` module. The default version will always point to the current point release. To continue the previous example, the default version when the 4.10.1 point release is created is also 4.10.1. If at some point a 4.10.2 point release needed to be created, then the default version would be switched to that version number upon approval by the key Sierra development leads.

As discussed in Section 6.1, the default version of Sierra is recommended for all situations that do not merit the use of a newer version of the code. If the same executable is required on a project for over one year (roughly the supported lifetime of a release executable), it is recommended that you copy the executable to ensure that it is not erased before you are done using it on the project.

6.2 End-of-Sprint (EOS) Snapshot

In between each major release, Sierra development occurs on odd-numbered development branches, e.g., 4.13. At the end of each three-week development sprint (See Chapter 7), the Sierra development teams also creates a point release, termed an End-of-Sprint (EOS) snapshot. EOS snapshots also follow the X.YY.Z convention, where Z is the sprint number in the current development cycle. These EOS snapshots are created to ensure that Sierra is release-ready at frequent intervals and to aid users who need to use a newer version than the current release.

As with release versions, Sierra development teams only support the two most recent EOS snapshots. This means that each EOS snapshot is available and supported for approximately six weeks.

6.3 Master

All Sierra development occurs on the development branch of the code, which is also termed `master` or Version of the Day (VOTD). To ensure code correctness, the `master` branch is built and tested nightly on all platforms upon which Sierra is supported. Sandia users may use the nightly-built version of Sierra by loading the following module:

```
$ module load sierra/master
```

Regular use of master is highly discouraged, as it is the least tested of all the versions of Sierra and may not be available due to regular code churn or system issues. Please exercise caution whenever using this version of Sierra. When running a Sierra application `master` executable, we recommend copying the executable to your disk area in case the executable becomes unavailable or you need to perform a restart run. The `sierra` script includes the `-c` option for this purpose.

6.4 Personal Versions

Some users may decide to participate in Sierra development in order to aid in personal projects. In this situation, the user will need to add the Sierra Developer account in Web-CARS (see Section 4.1). Once this account is approved, the Sierra `code` and `tests` Git repositories can be copied via the following commands. It is advised to do so on a fast filesystem such as `/scratch`:

```
$ module load sierra-devel
$ mkdir new_repo_dir
$ cd new_repo_dir
$ repo init
$ repo sync
$ repo start --all master
$ git clone sierra-git:/git/tests
```

To compile the code, the following command sequence will suffice:

```
$ module load sierra-devel
$ cd code
$ bake
```

Depending upon where the compilation occurs, it is wise to limit the number of processors to a reasonable amount. To compile one specific application, such as Sierra/SM, on 8 processors, execute the following:

```
$ bake -j 8 adagio
```

Once the executable has built, you can run any model with that binary:

```
$ module load sierra/master
$ sierra /path/to/code/bin/adagio -i input.i
```

For further information regarding Sierra development, please see the following URL:

http://compsim.sandia.gov/compsim/Docs/Sierra/development/index.html

Note that before a user engages in code development that will be deployed generally in Sierra code, they should engage with the correct code development team to ensure they are following practices which will result in high-quality code.

6.5 Distribution

The Sierra code is routinely distributed to external customers via processes that the Sierra DevOps team has created. Some internal users may also request a distribution if they feel that they have a compelling reason and are unable to utilize the supported Sierra versions on Sandia's HPC systems. In this case, a distribution request may be filled out at the following URL:

`http://compsim.sandia.gov/compsim/Support_SierraRequestForm.html`

Note that due to limited resources, these requests are generally discouraged if the Sierra code can be accessed in some other already established manner.

Chapter 7

Development Process

The Sierra development teams follow the concepts of lean/agile software development [11]. This development model was chosen in an effort to ensure that all deliverables are met while remaining flexible enough to meet changing demands in an ever-growing code development project.

7.1 SCRUM Agile Development

Most scrum development teams loosely adhere to the SCRUM development process, a widely used method that follows the principles of agile software development. More information regarding the scrum process can be found in the Scrum Guide [12]. Given that the scrum process is integral to interfacing with Sierra development teams, key information regarding the scrum process is described here. Realize that each development team is different and may employ practices which do not exactly align with the description below.

Development is broken into time-bound cycles called *sprints*, which last three weeks. Development tasks, called *User Stories*, are typically parts of larger efforts, called *epics*. Stories are usually sized so that they can be finished within a sprint. Every day during the sprint, the team will discuss progress and impediments at a 15-minute meeting called a *stand-up*. At the end of each sprint, the team will hold a one-hour *review* where they will present the progress on stories to any interested parties. They will then hold a team *retrospective* and *planning* session to discuss process improvements and plan tasks for the next sprint.

Several positions have been created within the scrum process to ensure success of the project. The *Product Owner* (PO) makes sure that the team achieves all deliverables by prioritizing all items on the *product backlog*, which generally consists of items from the fiscal year project plan, among other items. The *Scrum Master* is in charge of all meetings and helps to remove impediments to the team's productivity.

7.2 Process for Adding New Capabilities

There are several actions which may be taken to add a new capability to the Sierra code suite, each of which have varying levels of effectiveness. The method which ensures greatest success is to add a feature request to the fiscal year project plan, which is negotiated at the the management level. This requires a good deal of prior planning and negotiation and may not be feasible for some feature requests. The next most effective process is to communicate needs through the Product Owner so that they can place the request on the product backlog.

The least effective method for requesting a new feature is through the user support process, though this is also a viable option. Feature requests may be submitted to Sierra Help, who will then create an enhancement user support ticket. The time frame for completion of a feature request depends on the team and on the scope of the feature request. Smaller feature requests may be implemented quickly if the request fits in the with the current development work. More difficult enhancement requests may end up on the *ticket backlog*, which is then prioritized against the current set of user stories by the Product Owner. This is typically a good avenue to take when one has a low-priority request that should be recorded for future planning purposes. It is still recommended to consult with the correct Product Owner to ensure that they are aware of the feature request and its relative importance.

Appendices

Appendix A

Guide of Document Conventions

This appendix chapter outlines the different styles that are utilized in this manual. Styles and their respective descriptions are shown below.

Sandia-specific section

The entire section marked by the SNL Thunderbird logo is relevant only for Sandia users of Sierra.

Sandia-specific item(s)

Paragraphs/items marked by the Sandia Thunderbird logo are relevant only to Sandia users of Sierra. Examples are as follows:

- General item applicable to all users
- Sandia-specific item

This entire paragraph is applicable only to Sandia users of the Sierra codes. External users need not read this paragraph.

Command Line Prompt

```
$ cd /path/to/file
$ ls
```

Directory Listing

```
/path/to/important/directory
```

Text File

```
# File.txt

This is an example of a text file and how it appears
in this document.

BASH syntax highlighting is the default:

cd /path/to/file
for file in list ; do
    echo "Found $file"
done
```

Sierra Input File

```
$ Sierra example input file

Begin Sierra
    Begin Procedure
        Begin Time Control
            Time Step = 1.0
        End   Time Control

        Begin Region
            BC displacements on surface_1 = 0 0 0
        End   Region

    End   Procedure

End   Sierra
```

URL

http://www.sandia.gov/

Appendix B

Build Documentation

This appendix chapter describes how to obtain and build this document. The source code for this document is managed within Sierra's internal documentation repository. Appropriate WebCARS access will be required for this access. With access, the following command can be issued to obtain the source.

```
$ git clone sierra-git:/git/docs
```

This document's source resides within this directory:

```
$ cd docs/Sierra/user_manual
```

A brief introduction of the contents of this directory is given below.

`Makefile:`	Used to build the documentation; please read this file for its targets and their descriptions
`doc:`	Contains the LaTeX files and resultant final PDF document
`src:`	Contains the source file examples referenced within the document
`ref:`	Contains reference material used in the making of this document

Now, to build the document, the following command can be issued in the top-level directory.

```
$ make
```

Appendix C

Example Input File

Below is an example input file that showcases consistent indention and comments.

```
# <Problem_Description>
#
# time marching
# Dirichlet boundary condition
# Tabular Time-dependent Flux Boundary condition
# Tabular conductivity
# constant initial condition
# single material
#
# </Problem_Description>

Begin SIERRA Aria VOTD: Class-Model

  Title Aria Transient Training Model

$---------------------------------------------------
$ Define material properties to be used
$---------------------------------------------------
  Begin Aria Material ssteel304
    Density              = Constant rho = 7.900E+03 # kg/m^3, at 20 C
    Specific Heat        = Constant cp  = 477.0     # J/kg-K, at 20 C
    Heat Conduction      = Basic
    Thermal Conductivity = User_Function X = Temperature  Name =
   ssteel_k_function
  End   Aria Material ssteel304

  Begin Aria Material aluminum
    Density              = Constant rho = 2.770E+03 # kg/m^3, at 20 C
    Specific Heat        = Constant cp  = 800.0     # J/kg-K, at 20 C
    Heat Conduction      = Basic
    Thermal Conductivity = Constant k   = 175.0     # W/mK
  End   Aria Material aluminum
```

```

$---------------------------------------------------
$ Linear function for steel k vs Temperature
$---------------------------------------------------
  Begin Definition for Function ssteel_k_function
    Type is Piecewise Linear
    Begin Values
      #T (K)  k (W/mK)
      173.15  10.8857
      273.15  13.3978
      373.15  16.3285
      773.15  21.7714
      973.15  25.9582
      3500.0  25.9582
    End Values
  End   Definition for Function ssteel_k_function

$---------------------------------------------------
$ Linear function for unsteady heat flux
$---------------------------------------------------
  Begin Definition for Function unsteady_flux
    Type is Piecewise Linear
    Begin Values
      #t (s)  Q(W)
      0.0     0.0
      500.0   2.0e4
      4000.0  2.0e4
    End Values
  End   Definition for Function unsteady_flux

$---------------------------------------------------
$ Define linear solver settings
$ - Conjugate Gradient solver used
$ - Jacobi Preconditioning
$---------------------------------------------------
  Begin AZTEC Equation Solver solve_temperature_cg
    Solution Method          = CG
    Preconditioning Method   = Jacobi
    Maximum Iterations       = 500
    Residual Norm Tolerance = 1e-10
    Residual Norm Scaling    = ANORM # default
  End   AZTEC Equation Solver solve_temperature_cg

$---------------------------------------------------
$ Specify mesh name and settings
```

```
$-----------------------------------------------------
  Begin Finite Element Model FEModel
    Database Name = model_coarse_hex.g
#   Database Name = model_med_hex.g

$-----------------------------------------------------
$   Assign material properties to element blocks
$-----------------------------------------------------
    Use Material ssteel304 for \$
    block_1 block_4 block_5 block_6 block_7 block_8 \$
    block_9 block_10 block_21 block_22 block_31 \$
    block_32 block_33 block_120 block_121 block_122 block_123

$-----------------------------------------------------
$   Choose blocks to omit from analysis
$-----------------------------------------------------
    Omit Volume block_11

  End   Finite Element Model FEModel

$----------------------------------------------------------
$ Procedure domain - solution control, region settings
$----------------------------------------------------------

  Begin Procedure AriaProcedure

$-----------------------------------------------------
$   Define temporal solution parameters
$-----------------------------------------------------
    Begin Solution Control Description
      Use System Main
      Begin System Main
        Simulation Start Time            = 0.0
        Simulation Termination Time      = 3000.
        Begin Transient solution_block_1
          Advance AriaRegion
        End   Transient solution_block_1
      End   System Main

$-----------------------------------------------------
$     Specify time integration settings
$     Note: There could be several solution
$           blocks changing these parameters
$-----------------------------------------------------
      Begin Parameters for Transient Solution_Block_1
```

```
        Start Time        = 0.0     # seconds
        Termination Time = 3000.0 # seconds
        Begin Parameters for Aria Region AriaRegion
          Time Step Variation = Adaptive
          Initial Time Step Size = 0.05    # seconds
#         Initial Time Step Size = 0.004  # seconds
          Time Integration Method = Second_Order
          Maximum Time Step Size = 20.0    # seconds
#         Maximum Time Step Size = 4.0     # seconds
          Minimum Time Step Size = 0.01    # seconds
#         Minimum Time Step Size = 0.0001 # seconds
          Predictor-Corrector Tolerance = 0.01 # seconds
          Limit solution Increment Temperature = 20.0 # K
        End   Parameters for Aria Region AriaRegion
      End   Parameters for Transient Solution_Block_1
    End   Solution Control Description

$-------------------------------------------------------
$   Region domain - EQs, BCs, ICs, post-processing
$-------------------------------------------------------

    Begin Aria Region AriaRegion

$----------------------------------------------------
$     Define nonlinear solver parameters
$----------------------------------------------------
      Use Linear Solver solve_temperature_cg

      Nonlinear Solution Strategy  = Newton
      Maximum Nonlinear Iterations = 4
      Nonlinear Residual Tolerance = 1.e-6
      Nonlinear Relaxation Factor  = 1.0 # default

      # mimic Calore solution behavior
      Use DOF Averaged Nonlinear Residual
      Accept Solution After Maximum Nonlinear Iterations = True

$------------------------------------------------------------
$     Specify which mesh model to use for this region
$------------------------------------------------------------
      Use Finite Element Model FEModel

$----------------------------------------------------
$     Specify equations to solve
$----------------------------------------------------
```

```
      EQ Energy for Temperature on all_blocks using Q1 with Diff Mass

$---------------------------------------------------
$     Specify initial conditions
$---------------------------------------------------
      Begin Initial Condition IC_block
        Temperature = 300.0 # K
        All Volumes
      End   Initial Condition IC_block

$---------------------------------------------------
$     Specify boundary conditions
$---------------------------------------------------
      Begin Heat Flux Boundary Condition fluxBC
        Add Surface surface_701 surface_801
        Flux Time Function = unsteady_flux
      End   Heat Flux Boundary Condition fluxBC

      Begin Temperature Boundary Condition surface_temp_1
        Add Surface surface_202
        Temperature = 400. # K
      End   Temperature Boundary Condition surface_temp_1

$---------------------------------------------------
$     Define contents of binary plot file
$---------------------------------------------------
      Begin Results Output AriaOutput
        Database Name = transient.e
        At Step 0 Interval = 1
        Title Aria: Transient Training Model
        Nodal Variables = Solution->Temperature as T
      End   Results Output AriaOutput

    End   Aria Region AriaRegion

  End   Procedure AriaProcedure

End   SIERRA Aria VOTD: Class-Model
```

One way to properly indent Sierra input file source code is to use the `sindent` utility, developed by this document's authors. An example of how to use this utility is below. This utility may be used after loading a `sierra` module.

```
$ cat tst.i
#COMMENT ON COLUMN1
```

```
 bEgin block
this
#COMMENT ON COLUMN1
  bEgin block2
was
   End block2

#COMMENT ON COLUMN1
more
    bEgin block3
work
     bEgin block4
than
      End block4
 #COMMENT ON COLUMN2
it
       End block3
should've
                End block
been

$ sindent tst.i > tstnew.i
$ cat tstnew.i
#COMMENT ON COLUMN1
bEgin block
    this
#COMMENT ON COLUMN1
    bEgin block2
        was
    End block2

#COMMENT ON COLUMN1
    more
    bEgin block3
        work
        bEgin block4
            than
        End block4
        #COMMENT ON COLUMN2
        it
    End block3
    should've
End block
been
```

Appendix D

Example .bashrc and .bash_profile files

Below is an example .bashrc file. It is important to note that there are no modules loaded in the .bashrc as per the recommendations in Section 4.3.

```
#Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

export PATH=/usr/bin:/bin:/sbin:/usr/sbin:$PATH

export http_proxy=http://wwwproxy.sandia.gov:80

#MAKE PERMISSIONS MORE ACCESSIBLE
umask 027

#SET PATH
export PATH=${HOME}/bin:$PATH

#SET USER VARS
export MAILTO=${USER}@sandia.gov

#SET ALIAS
alias gochama='ssh -Y chama'
alias golynx='ssh -Y lynx'
alias back='cd $OLDPWD'
```

Below is an example .bash_profile.

```
# ~/.bash_profile: executed by bash(1) for login shells.
# see /usr/share/doc/bash/examples/startup-files for examples

# include .bashrc if it exists
if [ -f ~/.bashrc ]; then
    source ~/.bashrc
fi
```

Appendix E

Example .cshrc file

Below is an example .cshrc file. It is important to note that there are no modules loaded in the .cshrc as per the recommendations in Section 4.3.

```
if ( -r /usr/local/etc/system.cshrc ) then
  source /usr/local/etc/system.cshrc
endif
if ( -r /usr/local/modules/default/init/csh ) then
  source /usr/local/modules/default/init/csh
endif

# Set http proxy
setenv http_proxy http://wwwproxy.sandia.gov:80

limit coredumpsize 0k
umask 027

# list directories in columns
alias ls   'ls -C'
alias ll   'pwd ; ls -l'
alias lll  'pwd ; ls -al'
alias lld  'pwd ; ls -l  | grep drwx'
alias lad  'pwd ; ls -al | grep drwx'
alias llt  'pwd ; ls -lrt'

# useful aliases
alias ssh 'ssh -Y'

# Remember last 100 commands
set history = 100
alias h   'history'

# PATHS
set path = ( $HOME/bin /etc /usr/sbin /sbin /usr/kerberos/bin )
set path = ( $path /usr/local/etc /usr/local/bin /usr/bin /bin /usr/etc )
```

References

[1] Utkarsh Ayachit et al. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 4th edition, 2012. ISBN 978-1-930934-24-5.

[2] CEI, Inc. EnSight - CSM and CFD Post processing. https://www.ceisoftware.com, 2014.

[3] B. M. Adams, M. S. Ebeida, M. S. Eldred, J. D. Jakeman, L. P. Swiler, J. A. Stephens, D. M. Vigil, T. M. Wildey, W. J. Bohnhoff, K. R. Dalbey, J. P. Eddy, K. T. Hu, L. E. Bauman, and P. D. Hough. DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual. Technical report SAND2014-4633, Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, July 2014.

[4] Mahesh Rajan and Anthony Agelastos. Advantages & Benefits of 4GB/core Memory on Chama & Pecos. https://sharepoint.sandia.gov/sites/capviz/HPC%20User%20Meetings/memory_benefit_03.pdf, October 2013.

[5] Greg Sjaardema. SEACAS: Sandia Engineering Analysis Code Access System. http://compsim.sandia.gov/compsim/Docs/SEACAS/index.html, July 2013.

[6] Andrew C. Bauer, Berk Geveci, and Will Schroeder. *The ParaView Catalyst User's Guide v1.0*. Kitware Inc., 2013.

[7] Cubit Support. Sandia National Laboratories: CUBIT. https://cubit.sandia.gov/, 2014.

[8] Ernest Friedman-Hill, Edward Hoffman, Marcus Gibson, Kevin Olsen, et al. Sandia Analysis Workbench. https://dart.sandia.gov/wiki/display/SAW/Sandia+Analysis+Workbench, November 2014.

[9] Eric Harstad. Sandia National Laboratories: CTH Shock Physics. http://www.sandia.gov/CTH/index.html, 2015.

[10] Dakota Development Team. The Dakota Project: Large-scale Engineering Optimization and Uncertainty Analysis. http://dakota.sandia.gov, May 2014.

[11] Agile Development Team. Manifesto for Agile Software Development. http://agilemanifesto.org, 2001.

[12] Ken Schwaber and Jeff Sutherland. The SCRUM Guide$^{\text{TM}}$. http://www.scrumguides.org/scrum-guide.html, July 2013.

DISTRIBUTION:

1	MS	0807	Anthony M. Agelastos, 9326
1	MS	0807	Joel D. Miller, 9326
1	MS	0807	Ryan P. Shaw, 9326
1	MS	0845	Micheal W. Glass, 1545
1	MS	0899	Technical Library, 9536 (electronic copy)

v1.38

Sandia National Laboratories