

# SANDIA REPORT

SAND2019-3788

Printed April 4, 2019



Sandia  
National  
Laboratories

## SIERRA Code Coupling Module: Arpeggio User Manual – Version 4.52

Samuel R. Subia, James R. Overfelt, David G. Baur

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

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Telephone: (865) 576-8401  
Facsimile: (865) 576-5728  
E-Mail: [reports@osti.gov](mailto:reports@osti.gov)  
Online ordering: <http://www.osti.gov/scitech>

Available to the public from

U.S. Department of Commerce  
National Technical Information Service  
5301 Shawnee Road  
Alexandria, VA 22312

Telephone: (800) 553-6847  
Facsimile: (703) 605-6900  
E-Mail: [orders@ntis.gov](mailto:orders@ntis.gov)  
Online order: <https://classic.ntis.gov/help/order-methods>





## **ABSTRACT**

The SNL Sierra Mechanics code suite is designed to enable simulation of complex multiphysics scenarios. The code suite is composed of several specialized applications which can operate either in standalone mode or coupled with each other. Arpeggio is a supported utility that enables loose coupling of the various Sierra Mechanics applications by providing access to Framework services that facilitate the coupling. More importantly Arpeggio orchestrates the execution of applications that participate in the coupling. This document describes the various components of Arpeggio and their operability. The intent of the document is to provide a fast path for analysts interested in coupled applications via simple examples of its usage.



# CONTENTS

<b>Contents</b>	<b>5</b>
<b>List of Figures</b>	<b>8</b>
<b>1. Introduction</b>	<b>9</b>
1.1. Coupled Physics Approaches .....	9
1.2. Sierra Mechanics Coupling .....	10
1.3. Communication of Data (Transfer Services) .....	11
1.4. Solution Control .....	11
1.5. Coupling Strategies .....	12
1.6. Coupling with Arpeggio .....	15
1.7. Outline of the Manual .....	16
<b>2. Getting Started</b>	<b>17</b>
2.1. Setting The Environment-Users External to Sandia Labs .....	17
2.2. Setting The Environment-Users at Sandia Labs .....	17
2.3. Running Arpeggio .....	17
2.4. Arpeggio Environment Overview .....	18
2.5. Overview of the Input File Structure .....	19
2.6. Fields .....	25
2.7. User Fields .....	25
<b>3. Model Definition</b>	<b>26</b>
3.1. Model Overview .....	26
3.2. Finite Element Model .....	28
3.3. Parameters For Block .....	33

3.4.	Global Constants .....	36
3.5.	Definition For Function .....	39
3.6.	Values .....	48
3.7.	Restart Overview .....	49
<b>4.</b>	<b>Solution Control Reference</b>	<b>50</b>
4.1.	Overview .....	50
4.2.	Solution Control Description .....	60
4.3.	System .....	61
4.4.	Transient .....	65
4.5.	Nonlinear .....	70
4.6.	Subcycle .....	74
4.7.	Sequential .....	78
4.8.	Initialize .....	82
4.9.	Parameters For .....	83
<b>5.</b>	<b>Transfer Reference</b>	<b>92</b>
5.1.	Overview .....	92
5.2.	Transfer .....	98
<b>6.</b>	<b>Input Output Region Reference</b>	<b>108</b>
6.1.	Input_Output Region Overview .....	108
6.2.	Input_Output Region .....	109
<b>7.</b>	<b>Examples</b>	<b>114</b>
7.1.	One-Way Coupling From File .....	114
7.2.	One-Way Coupling Using Transfer From Different Mesh .....	118
7.3.	One-Way Coupling Using Transfer .....	119
7.4.	Two-Way Coupling With Transfer .....	125
7.5.	estack Regression Test .....	136

7.6. tv Regression Test .....	152
<b>Bibliography</b>	<b>157</b>
<b>Index</b>	<b>159</b>

# LIST OF FIGURES

1.1-1. Loose Coupling Schematic (Z Scheme). . . . .	10
1.3-1. Sierra Mechanics Data Types. . . . .	11
1.5-1. One-way Loose Coupling At Same Time Step. . . . .	13
1.5-2. Deferred One-way Loose Coupling At Same Time Step. . . . .	14
1.5-3. One-way Loose Coupling with Subcycling Schematic. . . . .	14
1.5-4. Two-way Loose Coupling Schematic. . . . .	15
1.6-1. Thermal-Mechanical With Thermal One-Way Element Death. . . . .	16
1.6-2. Thermal-Mechanical With Two-Way Element Death. . . . .	16
2.4-1. Schematic UML class diagram for the Expression subsystem. . . . .	19
5.1-1. Valid Transfer Operations . . . . .	93
5.1-2. Invalid Transfer Operation . . . . .	94

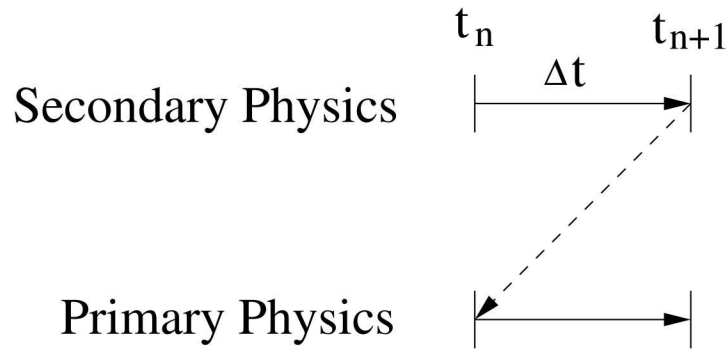
# 1. INTRODUCTION

The SNL Sierra Mechanics code suite is designed to enable numerical simulations of complex multi-physics scenarios. The code suite is composed of specialized applications which can operate either in standalone mode or in a coupled mode with other Sierra Mechanics applications. Arpeggio is a supported utility that enables loose coupling of the various Sierra Mechanics applications by providing access to Framework services that facilitate application coupling. Utilizing these services Arpeggio is able to orchestrate the execution of applications that participate in code coupling. This document describes the Framework services used by Arpeggio for coupling and the inter-operability of these services for coupling of Sierra SM and Sierra TF applications. Through the use of simple examples, the document also provides a resource for analysts interested in performing coupled-physics simulations.

## 1.1. COUPLED PHYSICS APPROACHES

When modelling tightly-coupled physics, the numerical representation of all PDEs within a region of interest are often combined a single system matrix and solved using a nonlinear solution strategy specific to the application. This approach to solving coupled-physics problems is available for a limited set of physics in the Sierra Mechanics TF module. Relaxing the notion of tight-coupling one could alternatively obtain solutions for each of the physics independently and patch the individual solutions together in some prescribed manner, this is the essence of loosely-coupled physics simulations.

The numerical analysis community has long recognized the need to include results from various physics in a single simulation. However, the fact that most application codes are often developed around single physics often limits the extent to which coupled-physics simulations can be achieved. Early approaches to coupled-physics simulations often simplified the coupling by level of importance by assigning primary physics and secondary physics roles. Here the primary physics depended upon secondary physics and the dependence of secondary physics upon primary physics was deemed less important. Under this assumption coupled physics simulations can be realized by first performing independent simulations of the secondary physics followed by a simulation of the primary physics utilizing results of the initial simulation. Figure 1.1-1 illustrates the coupling approach for a quasi-static solution step from a state  $t_n$  to state  $t_{n+1}$ . Broadly speaking, loose-coupling strategies are classified as Z-methods, since a Z describes the basic pattern of data communication between the physics applications. The one-way view of loosely-coupled physics lends itself to file-based approaches where single state results are obtained on a common spatially meshed discretization. Here the problem solutions are generally obtained at cell vertices (nodes) or cell centers (elements). Quite often each physics simulation lends itself to a particular spatial discretization and this gave rise to the introduction of an intermediate mapping step whereby the secondary physics results were mapped onto the primary physics discretization as in the MAPVAR utility [1]. For transient coupled-physics simulations best results are obtained when sharing a common



**Figure 1.1-1.. Loose Coupling Schematic (Z Scheme).**

time discretization but in many cases this is impractical and the coupling is based upon closest-time matched solutions or interpolations of solutions in time.

## 1.2. SIERRA MECHANICS COUPLING

Sierra Mechanics physics applications deal with solving PDEs on a physical geometric domain, a **Region**. In defining a coupled physics problem, users configure one or more **Regions** corresponding to some particular physics. Each **Region** considers one or more PDEs to be solved on either the entire input mesh or on a portion of the mesh. When the **Region** physics are coupled one can elect whether to solve the physics in a tightly-coupled manner in a single application or by loosely coupling individual **Region** results. Here we note that for loose coupling the physical geometry and spatial discretization must overlap but need not be identical in each of the participating **Regions**.

In the context of Sierra Mechanics, loosely-coupled physics nonlinear solutions are obtained on each of the **Regions** and then combined to form an overall coupled solution. Not surprisingly there are numerous ways one can approach loose-coupling since different strategies are appropriate to different problem sets. That is, the solution for one **Region** may depend strongly upon the solution in another **Region** but not vice-versa (one-way coupled), or the the solution for each **Region** may depend upon the solution the other **Region** (two-way coupling). The goal of Sierra Mechanics is to provide services which enable one to easily perform variants of a multi-physics coupling.

Some considerations which are relevant to the loose-coupling solution strategies include

- Communication of data from one **Region** to another **Region** (Transfer),
- Initialization of the individual **Regions**,
- Solution for the individual **Regions** (Advance),
- Time stepping or pseudo-time stepping for the individual **Regions**,
- Time synchronization of participating **Regions**,
- Conditional convergence,



- Drive mesh adaptivity,
- Sequencing for all of the above.

Within Sierra Mechanics communication of data between application **Regions** is handled by the Framework Transfer service and all aspects of solution behavior are managed by another Framework service, Solution Control. Mesh Adaptivity is managed through the Percept library.

## 1.3. COMMUNICATION OF DATA (TRANSFER SERVICES)

In Sierra Mechanics application data is generally associated with nodes, elements, faces or edges of a meshed discretization as shown in Figure 1.3-1. A loose-coupling between applications implies the dependence of one application on data supplied from some external source. Since the physical location of data on the external source may or may not map geometrically onto the other application solution, provisions must be made to perform this data mapping in a flexible manner. It is important to note that these mappings can be accomplished both for the case of different mesh and different element types. Within Sierra Mechanics this responsibility is handled by Framework Transfer services. Here it is important to note that Framework Transfer services enable the external source data to include element, face and edge data as well as nodal data.

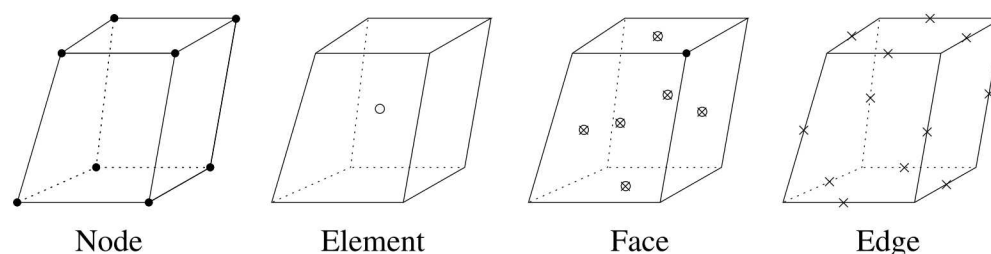


Figure 1.3-1.. Sierra Mechanics Data Types.

## 1.4. SOLUTION CONTROL

The Solution Control subsystem controls the execution of coupled multi-physics applications. Solution control provides two basic operations for controlling the solution of a multi-physics system by defining the order for object execution and by setting parametric values on the controlled objects at the proper time. For *transient* problems this approach enables the applications to easily transition through the designated time periods. The same system can also service steady-state simulations by treating them in a *sequential* manner. The solution controllers are able to initialize **Region** data, set parameter values, advance **Regions**, execute transfers, call events and send notifications based on the input file specifications.

### 1.4.1. Region Initialization

When beginning execution, all applications require some baseline initialization operations at the **Region** level. When performing some loose-coupling simulations the dependence of data may require that initialization of data be performed in some specific manner. Here the manner in which initialization occurs is determined by how the application solution variables are defined and the application code implementation of initialization. As an example for a thermal-mechanical coupling one might initialize the reference temperature state in the solid before any temperature change in the solid were allowed to occur. Solution Control provides a means for performing various types of non-standard data initialization.

### 1.4.2. Solution

Each set of coupled physics represents a *System* of equations which must be solved. While participating in loose-coupling an application physics will attempt to advance its solution to a later state. In the parlance of Solution Control this step is known as an advance event. Here the details of code operations associated with advancing the solution are controlled entirely by the physics application. Additionally, because the advance can occur conditionally it provides flexibility in how the coupling is performed.

### 1.4.3. Time Stepping

Within Sierra Mechanics each application is allowed to define its own notion of solution time. The Solution Control time step controller probes the individual application solution time and uses that information to determine how time should be advanced for the coupled physics. For couplings of transient simulations with quasi-static applications, the time step controller manages a unified notion of pseudo-time and physical time seamlessly, even when the time step selection is adaptive.

### 1.4.4. Conditional Events

In loosely-coupled simulations the need often arises to perform some high level operations conditionally. Here Solution Control is able to probe the application for current states or variables to determine whether some coupling action should occur. These conditionals can be applied to both the data transfer or advance Solution Control events. Examples of conditional events are included in Chapter [4](#).

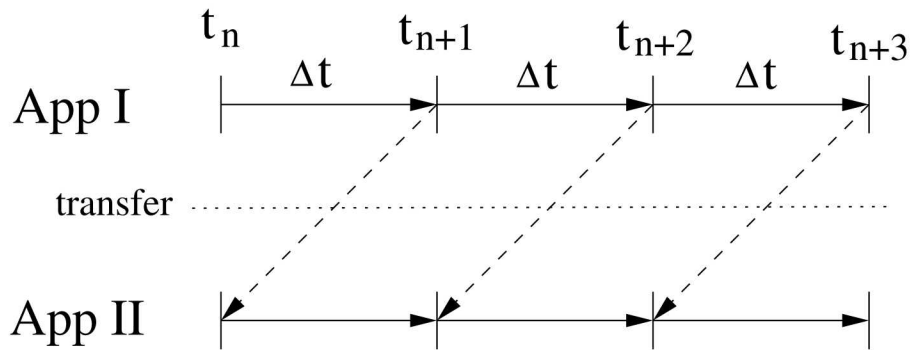
## 1.5. COUPLING STRATEGIES

Using the Solution Control one can easily define loose couplings between two or more **Regions**. For example, some or all of a solution from one **Region** may be transferred to another **Region** where it is treated as a constant, external field. The aggregate nonlinear problem including the contributions from all of the **Regions** may be iterated to convergence. The details of which physics are solved in each

**Region** and the nonlinear solution strategy used within and between **Regions** is completely specified through the input file. Furthermore, a Sierra Mechanics user may pick a simple, minimal algorithm without needing to fit it into an overly-generalized worst-case scenario that represents the union of all possible algorithms.

Dynamically-specified loose coupling has many potential advantages that users may leverage to obtain solutions. First, the resulting linear system is considerably smaller than a fully-coupled system and contains far fewer off-diagonal contributions which can significantly improve the performance of linear solvers. Furthermore the resulting linear system may have a more desirable mathematical properties, such as being symmetric positive-definite, this permits the use of tailored iterative solutions techniques. Other extensions to loose coupling include subcycling of transient simulations where each **Region** may advance in time with its own time step size and in-core coupling to other applications based upon the Sierra framework.

The simplest loose-coupling strategy is a one-way coupling between two applications, App I and App II, is shown schematically in Figure 1.5-1. Here it is assumed that information (data) from App I is needed by App II but App I is independent (decoupled) from App II. Furthermore it is assumed that the applications can proceed at the same time step. In this case the solution for each application can proceed in locked step.

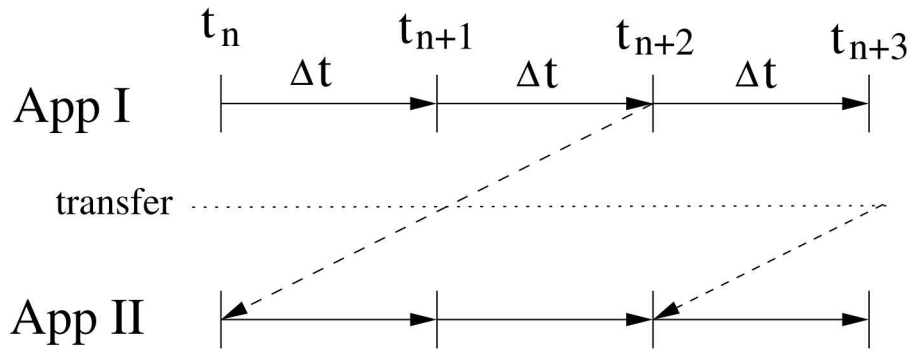


**Figure 1.5-1.. One-way Loose Coupling At Same Time Step.**

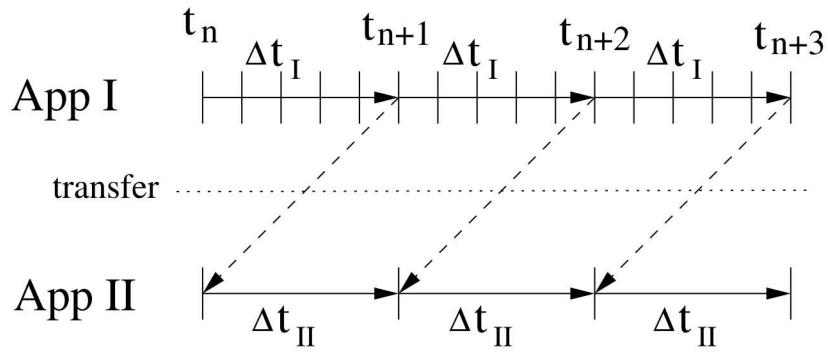
A variant of the simplest loose-coupling would be the case where the dependence of App II solutions on App I data is such that update of the App I data can be deferred for several steps. This type coupling behavior can be enforced using a conditional advance event in Solution Control. As an example, a data transfer event defined for every two time steps of each application is shown schematically in Figure 1.5-2.

In some couplings the temporal response of one application physics, App I, is much faster than that of another physics, App II. Here one may wish to advance the App I physics many time steps before requiring an update of its contribution to the App II information, Figure 1.5-3. Here Solution Control provides a facility denoted as *subcycling* to invoke this behavior.

When the coupling between App I and App II is circular in nature, (i.e. App I solutions depend upon App II and vice versa) the coupling can be achieved by adding an additional Transfer step to the one-way coupling approach. However, if the coupling dependency is fairly strong it may be prudent to ascertain a converged solution between the physics models before advancing to the solution step. Here the conditional event aspect of Solution Control can be employed to iterate App I and App II until a



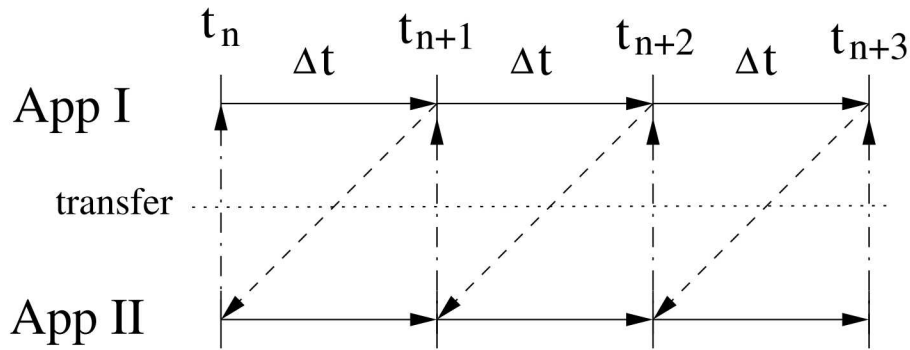
**Figure 1.5-2.. Deferred One-way Loose Coupling At Same Time Step.**



**Figure 1.5-3.. One-way Loose Coupling with Subcycling Schematic.**

solution of the desired quality is obtained. The strategy is depicted in Figure 1.5-4 and is supported as the Nonlinear option within Solution Control.





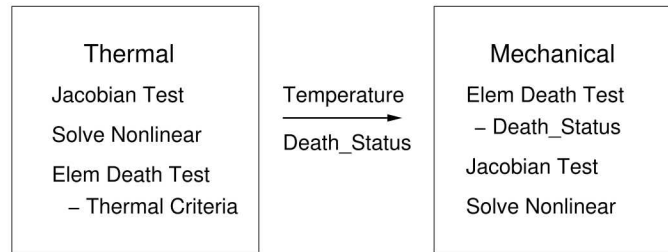
**Figure 1.5-4.. Two-way Loose Coupling Schematic.**

## 1.6. COUPLING WITH ARPEGGIO

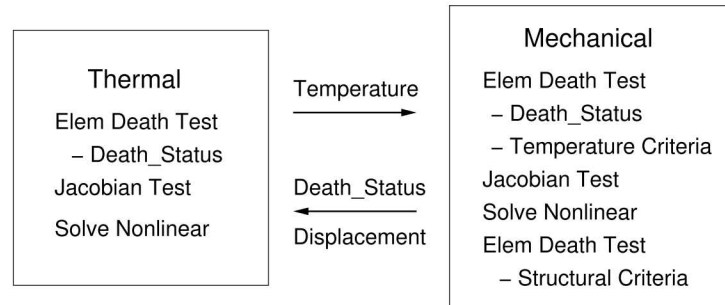
While the previous sections have described the component utilities needed to enable coupled physics simulations but little has been said of existing tools composed of these utilities. Previous efforts in the development of Sierra Mechanics focused upon thermal-mechanical coupling of the Calore and Adagio applications with the Calagio utility to analyze problems of thermal stress. Here Sierra Mechanics utilities were used to solve the temperature state, then initializing the reference temperature state in Adagio followed by subsequent solves and transfer of the temperature state to Adagio to obtain a thermal stress state in the deformed configuration. Within the Calagio utility extra efforts were made to obscure the use of Framework utilities lying outside the realm of Calore and Adagio. Early one-way coupling efforts were later followed by two-way couplings where the deformed configuration was communicated to Calore and the heat transfer problem could be solved in the updated geometry. Although couplings with Calagio were largely successful it was recognized that incremental improvements in coupling capability came with a high price in terms of code development effort both to alter the predefined coupling strategies and to hide the underlying implementation from the analyst within the application code. Moreover, the predefined coupling strategy approach prevented the analyst from fully exploiting the resources available within Sierra Mechanics and the applications themselves. These shortcomings provided a motivation for creation of the Arpeggio utility in which the analyst fully specifies details of the coupling strategy.

### 1.6.1. Coupling Including Element Death

Coupling strategies in predecessors of Arpeggio precluded the possibility of simulations that required synchronization of the meshed discretization such as element death. Here the transfer capability in conjunction with a consistent notion of an application code indicator of element death (Death\_Status) enables coupled simulations that include element death. Prevalent uses of this capability are one-way coupled thermal-mechanical simulations with thermally-driven element death [1.6-1](#) and two-way coupled thermal-mechanical simulations with element death driven by either application [1.6-2](#). For both types of coupling the mechanical code behavior is essentially the same as for a two-way coupling. On the other hand, in the case of two-way coupling the one-way coupling thermal invocation of a death criteria test is altered by the addition of an Aria Region level command line: Transfer Element Death.



**Figure 1.6-1.. Thermal-Mechanical With Thermal One-Way Element Death.**



**Figure 1.6-2.. Thermal-Mechanical With Two-Way Element Death.**

## 1.7. OUTLINE OF THE MANUAL

Chapter 2 discusses the overall Sierra Mechanics environment for running Arpeggio, including the layout for the Arpeggio input file. Sierra Mechanics users familiar with the overall environment need only browse the input file structure and move directly to the sections describing Framework Transfer 5 and Solution Control 3.7. Experienced Sierra Mechanics users may opt to move directly to examples of coupling in Chapter 7

## 2. GETTING STARTED

### 2.1. SETTING THE ENVIRONMENT-USERS EXTERNAL TO SANDIA LABS

To access Sierra/Arpeggio one will likely need to setup the user environment. This setup will differ upon location and the local system administrator can provide information on setting up your local environment.

### 2.2. SETTING THE ENVIRONMENT-USERS AT SANDIA LABS

The environment for using Arpeggio is the same as for individual Sierra applications and can be configured by module files. The modules ensure that the look and feel of running Sierra applications is the same across a multitude of compute platforms. To obtain the proper environment for code execution one simply runs:

```
% module load sierra
```

### 2.3. RUNNING ARPEGGIO

This section includes some very simple examples of how to run Arpeggio. For more information on running on some of Sandia's clusters, etc. see [\[2\]](#).

In its simplest form, Arpeggio can be run like this:

```
% sierra arpeggio -i myrun.i
```

In this example, `myrun.i` is the Arpeggio input file. The output – nonlinear iterations, time step information, etc. – will be written to a file called `myrun.log`. So, you can monitor the progress of the simulation by watching the log file. Alternatively, you can have all of the output sent to the display by using the `-l logfile` command line option. If you set the log file to be `-` (a single “minus” character) all of the output will be sent to the standard output (usually your display):

```
% sierra arpeggio -i myrun.i -l -
```

If you would like to use `aprepro` in your input file, add the `-a` command line option to have your input file automatically processed:

```
% sierra arpeggio -i myrun.i -l - -a
```

Oftentimes we want to run Arpeggio remotely or locally in a batch mode, save any standard output and perhaps even logout from a session. Unfortunately, termination of the session through either voluntary (interactive) or involuntary (timeout) logout may in effect terminate the Arpeggio job. In this case one can prevent the job from terminating by using the Unix `nohup` command in conjunction with the standard execution command line.

```
% nohup sierra arpeggio -i myrun.i -l YourLogFile -a
```

If one wishes to run the job in a background mode the `nohup` command should be terminated with `&` at the end of the command line.

## 2.4. ARPEGGIO ENVIRONMENT OVERVIEW

The Sierra Mechanics code suite is composed of several specialized applications which can operate either in standalone mode or coupled with each other. The various application models and algorithms are integrated into the Sierra framework through the architecture illustrated in Figure 2.4-1. A Sierra-based application has four layers of code: Domain, Procedure, Region, and Model/Algorithm.

The outermost layer of an application is the Domain, or “main” program of the application. This domain layer is implemented by the Sierra Framework to manage the startup/shutdown of an application, and to orchestrate the execution of an application-provided set of procedures.

Code at the Procedure level is responsible for evolving one or more loosely coupled set of physics through a sequence of steps. This sequence may be a set of time steps, nonlinear solver iterations, or some combinations of these or other types of steps.

An application may define multiple procedures to implement hand-off coupling between physics within the same main program. In hand-off coupling the first (or preceding) procedure completes execution, mesh and field data is transferred to a succeeding procedure, and the succeeding procedure continues the simulation with a different set of physics. For example, the first thermal procedure could calculate a temperature distribution inside a differentially heated fluid, and the second procedure could simulate natural convection of the fluid due to the density gradients set up by the resulting temperature field.

Code at the **Region** level is responsible for evolving a tightly coupled set of physics through a single step. Loose coupling of **Regions** is supported by the advanced transfer services provided by the Sierra framework.

Each **Region** owns (1) a set of models or algorithms that implement its tightly coupled set of physics and solvers and (2) an in-memory parallel distributed mesh and field database. This mesh and field data is fully distributed among parallel processors via domain decomposition.



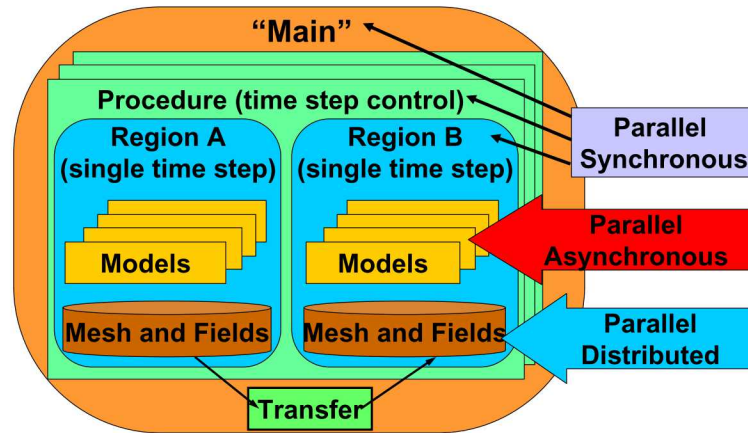


Figure 2.4-1.. Schematic UML class diagram for the Expression subsystem.

## 2.5. OVERVIEW OF THE INPUT FILE STRUCTURE

An Arpeggio model is described by commands contained in an ASCII input file. The structure of the input file follows a nested hierarchy. The topmost level of this hierarchy is named the domain. Below the domain lies a level named procedure, followed by the region level as depicted in Figure 2.4-1.

The domain level contains one or more procedures. At the domain level, one will also find commands associated with describing the finite element mesh, the linear solver set-up, material properties associated with a defined material, and user functions associated with source terms and boundary conditions that are added into Arpeggio’s intrinsic set of functions.

The procedure level contains one or more regions. The procedure level is also used to specify the time stepping parameters, and interactions between regions, such as data transfers. Essentially at the procedure level, loose coupling algorithms are specified. Loose coupling here is defined within the context of Arpeggio’s implicitly full-coupled paradigm. Whenever an independent variables’s interaction with other variables in the solution procedure is not fully represented in the global matrix, the algorithm for loose coupling of that variable and its associated equation will be described at the procedure level. This loose coupling algorithm is known as a “solution control description”. The procedure level contains a command block specifying the solution control procedure. An analogy to this block in simpler codes would be top level loop. For example in time dependent applications, the solution control description block would involve a block to solve the time dependent problem repeated for each time step until the desired solution time is reached.

The **Region** level is used to specify details about the physics to be solved. Details related to the solve

include boundary conditions and initial conditions, where materials models are applied, and where surface and volumetric source terms are applied. Here the meshed discretization and material properties described at the domain level are tied into the problem statement by virtue of their names.

Global constraint equations are also specified at the region level. At the region level, specification of information written to the output file and the frequency at which output occurs. Additional post-processing associated with the output is specified. For example, additional volumetric fields which are functions of the independent variables may be specified to be added to the output file.

There are two types of commands in the input file. The first type is referred to as a block command. A block command is a grouping mechanism. A block command contains a set of commands made up of other block commands and line commands. A line command is the second type of command. The domain, procedure, and region levels are all parsed as block commands. A block command is defined in the input file by a matching pair of Begin and End lines. For example,

```
Begin SIERRA myJob
... block commands
End SIERRA myJob
```

A set of key words for the block command follows the “Begin” and “End” keywords. In most cases a user-specified name is added to the block commands. In the example above the keywords, SIERRA myJob, are added. Optionally, the keyword may be left off of the end of the block.

The second type of command is the line command. A line command is used to specify parameters within a given block command. In the remaining chapters and sections of this manual, the scope of each block and line command is identified, along with summaries of the meanings. Note that the ordering of any commands within a command block is arbitrary. Thus,

```
Begin Finite Element model fluid
  Database name is pipeflow2d.g
  Use Material water for block_1
End Finite Element model fluid
```

will have the same effect as

```
Begin Finite Element model fluid
  Use Material water for block_1
  Database name is pipeflow2d.g
End Finite Element model fluid
```

And the ordering of command blocks within the domain/procedure/region blocks are arbitrary—allowing you considerable freedom to collect and arrange commands. Note that the terms “command block” and “block command” are interchangeable.

The Sierra command block must contain a block for a procedure containing at least one **Region**. For a case where only an Aria **Region** is being used:

```

Begin procedure  myProcedureName
.
  Begin Aria region myRegionName
.
  End  Aria region myRegionName
End procedure myProcedure

```

and similarly for a case using both Aria and Adagio Regions:

```

Begin procedure  myProcedureName
.
  Begin Adagio region myAdagioRegionName
.
  End  Adagio region myAdagioRegionName
.
  Begin Aria region myAriaRegionName
.
  End  Aria region myAriaRegionName
End procedure myProcedure

```

The procedure command block is used to contain all of the application code commands that are associated with a solution procedure defined for a set of **Regions**. The myProcedureName and name keywords of the procedure and region blocks are left up to you. Note that the procedure command block must be present in the input file and must contain at least one application code **Region** command block. The procedure command block also contains other important command blocks such as the SOLUTION CONTROL block.

## 2.5.1. Syntax Conventions for Commands

In this section we describe the conventions used in presenting all the command descriptions in the remainder of this manual. There are four basic kinds of tokens, or words, that an application code expects to find as it parses an input file. These are keywords, names, parameters and delimiters.

### 2.5.1.1. Keywords

The words which distinguish one block command, or line command, from another we term keywords. Keywords are denoted in this manual in the monospaced font, for example, BOUNDARY CONDITION.

### **2.5.1.2. Names**

The word, or words, that you supply on the same line of the begin line of a block command, is the name. Many times you may need to supply this name as a character parameter in a separate line command. Names are denoted in italics, name , as are parameters.

It is worth noting that the interpreter used to process standard input command lines is also used to process lines defining algebraic operations. This means that a "-" appearing within a name would be interpreted as a subtraction operation and as a consequence, the use of "-" within a name is not allowed. Thus instead of

```
Begin Adagio region name-1
```

one could perhaps use

```
Begin Adagio region name_1.
```

### **2.5.1.3. Parameters**

There are three types of input parameters one will need to supply to line commands: character strings, integers, and real numbers. These are denoted in the documentation as (C), (R), and (I), respectively.

In most cases character strings may be specified in a free format. One exception to this paradigm is when a string begins a number. In this case the character string must be specified within quotation marks in order to be properly interpreted.

Real numbers may be entered in decimal form or exponential form. For example 0.0001, .1E-3, 10.0d-5 are all equivalent. Furthermore, if a real(R) is expected, an integer can be used.

Integer values (I) need not include a decimal point in their specification.

### **2.5.1.4. Multiple Parameters**

For the case when a list of one or more parameters is allowed, or required, for a command, (C,...) denotes a list of character strings, (I,...) a list of integers, and (R, ...) a list of real numbers. For a list of character strings, the separator between the strings must be one or more spaces or tab characters. Therefore, phrases with multiple spaces and words in them are tokenized into multiple character parameters before being processed by the application. For a list of real or integer numbers the comma can also be used as a separator.

### **2.5.1.5. Enumerated Parameters**

Certain commands have predefined parameters, called enumerations, which are listed within {}. Each parameter in the list is separated using | . The default parameter for the list of parameters is enclosed by <>.

### **2.5.1.6. Delimiters**

The keywords of a line command are often required to be separated from the parameters by a delimiter. You have a choice of delimiters to use: the equal sign, =, or a word. In this manual, we denote the choices surrounded by {}, and separated by |. You may use any one of the delimiters from those listed. For example, the line command to specify the density within the Calore Material Block command is

Density {= |IS} (R)

Examples of valid forms you could write in the input file are

Begin Property Specification for Calore Material water

```
...
Density = 1.0E-3 # kg/m\^3 at 20C
...
End
```

and

Begin Aria Material water

```
...
Density is constant rho = 1.0E-3 # kg/m\^3 at 20C
...
End
```

### **2.5.1.7. White Space**

Command keywords, names, and parameters and delimiters must have spaces around them.

### **2.5.1.8. Indentation**

All leading spaces and/or tab characters are ignored in the input file. Of course, we recommend that you use indentation to improve the readability for yourself and others that may need to see your files.

### **2.5.1.9. Case Sensitivity**

None of the command keywords, parameters, or delimiters read from the input file are case sensitive. For example, the following two lines are equivalent:

```
Use Material water for block_1
```

```
.
```

and

```
USE material wATer for BLOCK_1
```

```
.
```

The exception to this rule are file names used for input and output, because the current operating systems on which SIERRA applications are run are based on UNIX, where file names are case sensitive.

#### **2.5.1.10. Comments and Line Continuation**

You may place comments in the input file starting with either the \$ or # character. All further characters on a line following a comment character are ignored.

You can continue a command in the input file to the next line by using the line continuation character \$, or you may optionally following it with a comment#. All further characters on the same line following a line continuation character \$ are ignored, and the characters on the following line are joined and parsing continues. An example is the line command used to specify the title of a thermal model:

```
Begin SIERRA Job_Identifier
# This thermal model for Calore simulates a convective heat transfer

Title   The title command is used to set the analysis title $\
        Convective heat transfer to a part. The analysis $\
        makes use of conjugate heat transfer to account for  $\
        cooling of a part due to flowing water.
...
End SIERRA Job_Identifier
```

#### **2.5.1.11. Checking the Syntax**

Errors in the input deck can be checked by adding the command, “--check-syntax” to the aria command line. For example,

```
sierra arpeggio --check-syntax -i input.i
```

This command will print the code echo of the input deck and any syntax errors within it to the display.

## 2.6.    **FIELDS**

Fields are defined as variables which are distributed on mesh objects (e.g. nodes, elements, faces or edges). The mesh object and Field data may be distributed among parallel processors via a domain decomposition algorithm. Each application registers Fields by name on its own **Region**. In a coupled-physics simulation Framework transfer services may be called on to communicate these Fields to another application. For example, the temperature Field in one application may be communicated to a solid mechanics application in order to perform a thermal-stress analysis.

## 2.7.    **USER FIELDS**

Situations often arise where one wishes to provide Field data storage so that data can be transferred into or out of the application. Each of the application codes provide some mechanism for enabling this type of data access. Additionally, User Fields are often used to as additional storage needed in user supplied subroutines.

## 3. MODEL DEFINITION

### 3.1. MODEL OVERVIEW

Sierra Framework services provide overall control of input commands, discretization input data and output data, IO. Additionally they provide a directed interaction of Framework services at the so-called Domain level with the application code at the Region level. This controlled interaction is enabled by commands that follow.

The model discretization (mesh) and the mesh components to be used in the model are defined at the Domain level and are later referenced by the application at the Region level. The association of material properties with portions of the mesh are also defined here within the Finite Element Model command block/s. For some couplings using the same mesh a single Finite Element Model may be used but for most cases one will use separate Finite Element Model command blocks for each **Region**. A sample outline of a setup for coupling of a solid mechanics application sm and a thermal-fluid tf is shown below.

```
Begin Sierra myJob
.
Begin Finite Element Model my_fem_model_sm
.
End
.
Begin Finite Element Model my_fem_model_tf
.
End

Begin Global Constants
.
End
.
Model definition commands

- Material definitions for sm
- Function definitions for sm
- Local Coordinate Systems for sm
.
- Material definitions for tf
```



- Function definitions for  $tf$
- Local Coordinate Systems for  $tf$

.

```

Begin Procedure My_Procedure
.
procedural commands
- Solution Control Description
- Transfer operations
.
Begin Adagio Region My_Adagio_Region
.
use Finite Element Model my_fem_model_sm
.
- sm Region level commands
.
End
.
Begin Aria Region My_Aria_Region
.
use Finite Element Model my_fem_model_tf
.
- tf Region level commands
.
End
.
End
End Sierra myJob

```

Note that a given application may not support the entire set of available options available in the Finite Element Model command block, particularly in the Parameters for Block section. Rather than attempting to include the entire set of command lines available in the Finite Element command block, only a small subset of key command lines are shown here. One should consult documentation for the specific application to find a complete listing of the relevant Finite Element Model command lines.

## 3.2. FINITE ELEMENT MODEL

Scope: Sierra

---

```

Begin Finite Element Model Label

Alias DatabaseName As InternalName

Component Separator Character Option Separator

Create GroupType NewSurfaceName Add SurfaceName...

Coordinate System {=|are|is} CoordinateSystem

Database Name {=|are|is} StreamName

```

```

Database Type {=|are|is} DatabaseTypes
Decomposition Method {=|are|is} Method
Global Id Mapping Backward Compatibility Option1 Option2
Omit Block BlockList...
Omit Volume VolumeList...
Time Scale Factor Option Scale
Use Generic Names
Use Material MaterialName For VolumeList...
Begin Parameters For Block Blockname
End

Begin Parameters For Phase Phase Name
End

Begin Parameters For Surface Surface_Name
End

End

```

---

**Summary** Describes the location and type of the input stream used for defining a geometry model for the enclosing region.

### 3.2.1. Alias

Scope: Finite Element Model

---

```
Alias DatabaseName As InternalName
```

Parameter	Value	Default
<i>DatabaseName</i>	string	undefined
<i>InternalName</i>	string	undefined

---

**Summary** Name the database entity "DatabaseName" as "InternalName"

**Description** This "InternalName" may then be referenced in the data file in addition to the original name.

### 3.2.2. Component Separator Character

Scope: Finite Element Model

---

Component Separator Character *Option Separator*

Parameter	Value	Default
<i>Option</i>	{=   is}	undefined
<i>Separator</i>	string	undefined

**Summary** The separator is the single character used to separate the output variable basename (e.g. "stress") from the suffices (e.g. "xx", "yy") when displaying the names of the individual variable components. For example, the default separator is "\_", which results in names similar to "stress\_xx", "stress\_yy", ... "stress\_zx". To eliminate the separator, specify an empty string ("") or NONE.

### 3.2.3. Create

Scope: Finite Element Model

---

Create *GroupType NewSurfaceName Add SurfaceName...*

Parameter	Value	Default
<i>GroupType</i>	{edgeset   elemset   faceset   nodeset   sideset   surface}	undefined
<i>NewSurfaceName</i>	string	undefined
<i>SurfaceName</i>	string...	undefined

**Summary** Create a new set (node, edge, face, element, side/surface) as the union of two or more existing sets. The sets must exist in the mesh database or have been created by a previous CREATE command.

### 3.2.4. Coordinate System

Scope: Finite Element Model

---

Coordinate System {= | are | is} *CoordinateSystem*

Parameter	Value	Default
<i>CoordinateSystem</i>	{axisymmetric   barycentric   cartesian   cyclidic   cylindrical   polar   quadriplanar   skew   spherical   toroidal   trilinear}	undefined

**Summary** The interpretation of the geometry data stored in this database. Optional. Defaults to Cartesian.

### 3.2.5. Database Name

Scope: Finite Element Model

---

Database Name {=|are|is} *StreamName*

Parameter	Value	Default
<i>StreamName</i>	string	undefined

**Summary** The base name of the database containing the output results. If the filename begins with the '/' character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a ".g" suffix appended.

### 3.2.6. Database Type

Scope: Finite Element Model

---

Database Type {=|are|is} *DatabaseTypes*

Parameter	Value	Default
<i>DatabaseTypes</i>	{catalyst   dof   dof_exodus   exodus   exodusii   generated   genesis   parallel_exodus}	undefined

**Summary** The database type/format used for the mesh.

### 3.2.7. Decomposition Method

Scope: Finite Element Model

---

Decomposition Method {=|are|is} *Method*

Parameter	Value	Default
<i>Method</i>	{block   cyclic   external   geom_kway   hsfc   kway   kway_geom   linear   metis_sfc   random   rcb   rib}	undefined

**Summary** The decomposition algorithm to be used to partition elements to each processor in a parallel run.

### 3.2.8. Global Id Mapping Backward Compatibility

Scope: Finite Element Model

---

Global Id Mapping Backward Compatibility *Option1 Option2*

Parameter	Value	Default
<i>Option1</i>	{=   is}	undefined
<i>Option2</i>	{false   no   off   on   true   yes}	undefined

**Summary** (Unsupported, do not use)

### 3.2.9. Omit Block

Scope: Finite Element Model

---

Omit Block *BlockList...*

Parameter	Value	Default
<i>BlockList</i>	string...	undefined

**Summary** Specifies that the element blocks named in the blockList be omitted from the analysis.

**Description** If an element block is omitted, then it is illegal to refer to it later in the input file e.g an initial condition may not be specified on an omitted element block. The elements, faces, etc are never created and it is as if the omitted element blocks did not exist in the mesh file. If a surface is completely determined by the omitted element block, then it is illegal to specify boundary conditions on that surface. However, if the surface spans multiple element blocks, boundary conditions may be applied on the portion of the surface supported by the element blocks that are not omitted.

### 3.2.10. Omit Volume

Scope: Finite Element Model

---

Omit Volume *VolumeList...*

Parameter	Value	Default
<i>VolumeList</i>	string...	undefined

**Summary** Specifies that the volumes named in the volumeList be omitted from the analysis.

**Description** If a volume is omitted, then it is illegal to refer to it later in the input file e.g an initial condition may not be specified on an omitted volume. The elements, faces, etc are never created and it is as if the omitted volumes did not exist in the mesh file. If a surface is completely determined by the omitted volume, then it is illegal to specify boundary conditions on that surface. However, if the surface spans multiple volumes, boundary conditions may be applied on the portion of the surface supported by the volumes that are not omitted.

### 3.2.11. Time Scale Factor

Scope: Finite Element Model

---

Time Scale Factor *Option Scale*

Parameter	Value	Default
<i>Option</i>	{=   is}	undefined
<i>Scale</i>	real	undefined

**Summary** The scale factor to be applied to the times on the mesh database. If the scale factor is 20 and the times on the mesh database are 0.1, 0.2, 0.3, then the application will see the mesh times as 2, 4, 6.

### 3.2.12. Use Generic Names

Scope: Finite Element Model

**Summary** If this command is present then the name of all blocks and sets in the mesh will be of the form "type\_" + id. For example, an element block with id=42 will be named "block\_42"; a sideset with id 314 will be named "surface\_314". If there are any names in the mesh file, those names will be aliases for the blocks and sets. If this command is not present, then if a name is in the mesh file, it will be used as the name and the generic generated name will be an alias. This is used as a workaround in codes that do not correctly handle named blocks and sets or as a workaround in meshes which contain non-user-specified names.

### 3.2.13. Use Material

Scope: Finite Element Model

---

Use Material *MaterialName* For *VolumeList...*

Parameter	Value	Default
<i>MaterialName</i>	string	undefined
<i>VolumeList</i>	string...	undefined

**Summary** Associate the given volumes with the indicated material name.

## 3.3. PARAMETERS FOR BLOCK

Scope: Finite Element Model

---

Begin Parameters For Block *Blockname*

Include All Blocks

Inversion Aversion Exponent {= | are | is} *ia\_exponent*

Inversion Aversion Stiffness {=|are|is} *ia\_stiffness*

Inversion Aversion Transition Jacobian {=|are|is}  
*transition\_jacobian*

Local Coordinate System {=|are|is} *Mesh Entities*

Material *MatName*

Material = *MatName*

Phase *PhaseLabel* {=|are|is} *MaterialName*

Remove Block {=|are|is} *ExcludeBlockList...*

End

---

**Summary** Specifies analysis parameters associated with each element block.

### 3.3.1. Include All Blocks

Scope: Parameters For Block

**Summary** Use this parameters definition for all blocks.

When using this option within the FINITE ELEMENT MODEL command block the PARAMETERS FOR BLOCK will not use a Blockname.

### 3.3.2. Inversion Aversion Exponent

Scope: Parameters For Block

---

Inversion Aversion Exponent {=|are|is} *ia\_exponent*

Parameter	Value	Default
<i>ia_exponent</i>	integer	5

---

**Summary** Sets the exponent used to compute the smooth approximate nodal jacobian ratio. A higher exponent results in a more-accurate approximation to the ratio. This is only active for uniform gradient elements. Default = 5.



### 3.3.3. Inversion Aversion Stiffness

Scope: Parameters For Block

---

Inversion Aversion Stiffness {=|are|is} *ia\_stiffness*

Parameter	Value	Default
<i>ia_stiffness</i>	real	1.e5

**Summary** Sets a stiffness parameter for the inversion aversion penalty. This is only active for uniform gradient elements. Default = 1.0e5.

### 3.3.4. Inversion Aversion Transition Jacobian

Scope: Parameters For Block

---

Inversion Aversion Transition Jacobian {=|are|is} *transition\_jacobian*

Parameter	Value	Default
<i>transition_jacobian</i>	real	0

**Summary** Sets the critical relative nodal Jacobian ratio for inversion aversion. If this value is nonzero, an additional recoverable energy term is added which penalizes further element distortion. This energy is only active for uniform gradient elements.

### 3.3.5. Local Coordinate System

Scope: Parameters For Block

---

Local Coordinate System {=|are|is} *Mesh Entities*

Parameter	Value	Default
<i>Mesh Entities</i>	string	undefined

**Summary** Associate coordinate system with mesh entity.

**Description** Specify the local coordinate system to be used in conjunction with given element blocks.

### 3.3.6. Material

Scope: Parameters For Block

---

Material *MatName*

Parameter	Value	Default
<i>MatName</i>	string	undefined

**Summary** Associates this element block with its material properties.

### 3.3.7. Material =

Scope: Parameters For Block

---

Material = *MatName*

Parameter	Value	Default
<i>MatName</i>	string	undefined

---

Summary Associates this element block with its material properties.

### 3.3.8. Phase

Scope: Parameters For Block

---

Phase *PhaseLabel* {=|are|is} *MaterialName*

Parameter	Value	Default
<i>PhaseLabel</i>	string	undefined
<i>MaterialName</i>	string	undefined

---

Summary Associate phase *PhaseLabel* with material *Material\_Name* on this block.

### 3.3.9. Remove Block

Scope: Parameters For Block

---

Remove Block {=|are|is} *ExcludeBlockList...*

Parameter	Value	Default
<i>ExcludeBlockList</i>	string...	undefined

---

Summary List of blocks to exclude.

## 3.4. GLOBAL CONSTANTS

Scope: Sierra

---

Begin Global Constants *empty*

Gravity Vector {=|are|is} *Gravity<sub>1</sub>* *Gravity<sub>2</sub>* *Gravity<sub>3</sub>*

Ideal Gas Constant {=|are|is} *Sigma*

K-E Turbulence Model Parameter *Param* {=|are|is} *Value*

K-W Turbulence Model Parameter *Param* {=|are|is} *Value*

Les Turbulence Model Parameter *Param* {=|are|is} *Value*

Light Speed {=|are|is} *LightSpeed*

Planck Constant  $\{=|are|is\}$  *PlanckConstant*  
 Stefan Boltzmann Constant  $\{=|are|is\}$  *Sigma*  
 Turbulence Model *Param* Number  $\{=|are|is\}$  *Value*

End

---

**Summary** Set of universal constants for a simulation.

### 3.4.1. Gravity Vector

Scope: Global Constants

---

Gravity Vector  $\{=|are|is\}$  *Gravity<sub>1</sub> Gravity<sub>2</sub> Gravity<sub>3</sub>*

Parameter	Value	Default
<i>Gravity</i>	real_1 real_2 real_3	undefined

---

**Summary** Gravity constant in vector form, acceleration components.

### 3.4.2. Ideal Gas Constant

Scope: Global Constants

---

Ideal Gas Constant  $\{=|are|is\}$  *Sigma*

Parameter	Value	Default
<i>Sigma</i>	real	undefined

---

**Summary** Ideal gas constant. **NOTE:** Another ideal gas constant value can be specified while using certain code capabilities. This global constants value will be discarded for any other specified ideal gas constant values.

### 3.4.3. K-E Turbulence Model Parameter

Scope: Global Constants

---

K-E Turbulence Model Parameter *Param*  $\{=|are|is\}$  *Value*

Parameter	Value	Default
<i>Param</i>	string	undefined
<i>Value</i>	real	undefined

---

**Summary**  $k - \epsilon$  RANS turbulence model parameters.

### 3.4.4. K-W Turbulence Model Parameter

Scope: Global Constants

---

K-W Turbulence Model Parameter *Param* {=|are|is} *Value*

Parameter	Value	Default
<i>Param</i>	string	undefined
<i>Value</i>	real	undefined

---

Summary  $k - \omega$  RANS turbulence model parameters.

### 3.4.5. Les Turbulence Model Parameter

Scope: Global Constants

---

Les Turbulence Model Parameter *Param* {=|are|is} *Value*

Parameter	Value	Default
<i>Param</i>	string	undefined
<i>Value</i>	real	undefined

---

Summary LES turbulence model parameters.

### 3.4.6. Light Speed

Scope: Global Constants

---

Light Speed {=|are|is} *LightSpeed*

Parameter	Value	Default
<i>LightSpeed</i>	real	undefined

---

Summary Speed of Light. Depending on the units involved in the specific problem by the user, this value will differ.

### 3.4.7. Planck Constant

Scope: Global Constants

---

Planck Constant {=|are|is} *PlanckConstant*

Parameter	Value	Default
<i>PlanckConstant</i>	real	undefined

---

Summary Planck Constant. Depending on the units involved in the specific problem by the user, this value will differ.

### 3.4.8. Stefan Boltzmann Constant

Scope: Global Constants

---

Stefan Boltzmann Constant {=|are|is} *Sigma*

Parameter	Value	Default
<i>Sigma</i>	real	undefined

**Summary** Stefan-Boltzmann constant. Depending on the units involved in the specific problem by the user, this value will differ.

### 3.4.9. Turbulence Model

Scope: Global Constants

---

Turbulence Model *Param* Number {=|are|is} *Value*

Parameter	Value	Default
<i>Param</i>	string	undefined
<i>Value</i>	real	undefined

**Summary** Turbulence model Schmidt and Prandtl numbers

## 3.5. DEFINITION FOR FUNCTION

Scope: Sierra

---

Begin Definition For Function *FunctionName*

Abscissa {=|are|is} *Name...*

Abscissa Offset {=|are|is} *Abscissa\_offset*

Abscissa Scale {=|are|is} *Abscissa\_scale*

At Discontinuity Evaluate To *Option*

Column Titles *Titles<sub>1</sub> Titles<sub>2</sub>...*

Data File = *filename* [ X From Column *xcol* Y From Column *ycol* ]

Debug {=|are|is} *Option*

Differentiate Expression {=|are|is} *Expr*

Evaluate Expression {=|are|is} *Expr*

Evaluate From *x0* To *x1* By *Dx*

Expression Variable: *Expr* = *VarType value\_var\_name...*

Expression Variable: *Expr*

```

Ordinate {=|are|is} Name...

Ordinate Offset {=|are|is} Ordinate_offset

Ordinate Scale {=|are|is} Ordinate_scale

Scale By x

Type {=|are|is} Type

X Offset {=|are|is} X_offset

X Scale {=|are|is} X_scale

Y Offset {=|are|is} Y_offset

Y Scale {=|are|is} Y_scale

Begin Expressions empty

End

Begin Values empty

End

```

End

---

**Summary**    Defines a function in terms of its type and values.

### 3.5.1.    **Abscissa**

Scope:    Definition For Function

---

```
Abscissa {=|are|is} Name...
```

Parameter	Value	Default
<i>Name</i>	string...	undefined

---

**Summary**    Specifies a string identifier for the independent variable. Optionally specify a scale and/or offset value which transforms the abscissa values into  $\text{scaled\_abscissa} = \text{scale} * (\text{abscissa} + \text{abscissa\_offset})$ .

### 3.5.2. Abscissa Offset

Scope: Definition For Function

---

Abscissa Offset {=|are|is} *Abscissa\_offset*

Parameter	Value	Default
<i>Abscissa_offset</i>	real	undefined

Summary Alias for X OFFSET

### 3.5.3. Abscissa Scale

Scope: Definition For Function

---

Abscissa Scale {=|are|is} *Abscissa\_scale*

Parameter	Value	Default
<i>Abscissa_scale</i>	real	undefined

Summary Alias for X SCALE

### 3.5.4. At Discontinuity Evaluate To

Scope: Definition For Function

---

At Discontinuity Evaluate To *Option*

Parameter	Value	Default
<i>Option</i>	{left right}	undefined

Summary Control the behavior of a piecewise constant function when evaluated at a discontinuity (plus or minus a small tolerance). The default behavior is to take the value to the right of the discontinuity. If "Left" is specified, the value to the left of the discontinuity is taken instead.

### 3.5.5. Column Titles

Scope: Definition For Function

---

Column Titles *Titles<sub>1</sub> Titles<sub>2</sub>...*

Parameter	Value	Default
<i>Titles</i>	string_1 string_2...	undefined

Summary Name the columns (and also defined the expected number of columns) for Multicolumn Piecewise Linear tabular data.

### 3.5.6. Data File

Scope: Definition For Function

---

Data File = *filename* [ X From Column *xcol* Y From Column *ycol* ]

Parameter	Value	Default
<i>filename</i>	string	undefined

**Summary** Function will read tabular data from an input file. Compatible with the piecewise linear function type. File must be of form like:

```
----- # EXAMPLE FILE 1.099 1191 1.101 221 5.9011 133.1
-----
```

Lines headed by a # are considered comments and will be ignored. Data itself must be in tabular columns separated by whitespace or commas.

### 3.5.7. Debug

Scope: Definition For Function

---

Debug {=|are|is} *Option*

Parameter	Value	Default
<i>Option</i>	{off   on}	undefined

**Summary** Prints functions to the log file.

### 3.5.8. Differentiate Expression

Scope: Definition For Function

---

Differentiate Expression {=|are|is} *Expr*

Parameter	Value	Default
<i>Expr</i>	(expression)	undefined

**Summary** Specifies the expression of derivative of evaluation expression.



### 3.5.9. Evaluate Expression

Scope: Definition For Function

---

Evaluate Expression {=|are|is} *Expr*

Parameter <i>Expr</i>	Value (expression)	Default undefined
--------------------------	-----------------------	----------------------

---

**Summary** Specifies the expression to evaluate.

**Description** This will greatly help with manufactured solutions, and be useful for other purposes as well.

This first implementation goes like this:

```
begin definition for function pressure
type is analytic
evaluate expression is "x <= 0.0 ? 0.0 : (x < 0.5 ?
x*200.0 : (x < 1.0 ? (x - 0.5) *50.0 + 100.00 :
150.0));"
# type is piecewise linear
# begin values
# 0.0 0.0
# 0.5 100.0
# 1.0 150.0
# end values
end definition for function pressure
```

Also, notice that semicolon at the end. Be sure to put it there for now. You can actually provide multiple expressions to be evaluated, each terminated with a semicolon. This will be handy when multi-dependent variable come into the fold.

The following functions are currently implemented.

**Operators** All C-language operators are supported, e.g. + - \*/ || ? : etc

**Parens** ()

**Math Functions**

**abs(x)** absolute value of x

**mod(x, y)** modulus of x|y

**ipart(x)** integer part of x

**fpart(x)** fractional part of x

**min(x0, x1, ...)** minimum value of xn

**max(x0, x1, ...)** maximum value of xn

## Power functions

**pow(x, y)** x to the y power

**sqrt(x)** square root of x

## Trig functions

**sin(x)** sine of x

**sinh(x)** hyperbolic sine of x

**asin(x)** arcsine of x

**cos(x)** cosine of x

**cosh(x)** hyperbolic cosine of x

**acos(x)** arccosine of x

**tan(x)** tangent of x

**tanh(x)** hyperbolic tangent of x

**atan(x)** arctangent of x

**atan2(y, x)** arctangent of y/x, signs of x and y determine quadrant (see atan2 man page)

## Logarithm functions

**log(x)** natural logarithm of x

**ln(x)** natural logarithm of x

**exp(x)** e to the x power

**logn(x, y)** the y base logarithm of x

## Rounding functions

**ceil(x)** smallest integral value not less than x

**floor(x)** largest integral value not greater than x

## Random functions

**rand(x)** random number between 0.0 and 1.0, not including 1.0

**srand(x)** seeds the random number generator

## Conversion routines

**deg(x)** converts radians to degrees

**rad(x)** converts degrees to radians

**recttopolr(x, y)** magnitude of vector x, y

**recttopola(x, y)** angle of vector x, y

**poltorectx(r, theta)** x coordinate of angle theta at distance r

**poltorecty(r, theta)** y coordinate of angle theta at distance r

### 3.5.10. Evaluate From

Scope: Definition For Function

---

Evaluate From  $x0$  To  $x1$  By  $Dx$

Parameter	Value	Default
$x0$	real	undefined
$x1$	real	undefined
$Dx$	real	undefined

---

**Summary** Specifies the range and evaluation interval.

### 3.5.11. Expression Variable:

Scope: Definition For Function

---

Expression Variable: *Expr* = *VarType* *value\_var\_name*...

Parameter	Value	Default
<i>Expr</i>	string	undefined
<i>VarType</i>	{element   element_sym_tensor   element_tensor   element_vector   face   global   nodal   nodal_sym_tensor   nodal_tensor   nodal_vector}	undefined
<i>value_var_name</i>	string...	undefined

---

**Summary** Specifies what the arguments of an expression correspond to. For example:

```
BEGIN DEFINITION FOR FUNCTION dx_shear TYPE = ANALYTIC
EXPRESSION variable: mx = NODAL model_coordinates(x) EXPRESSION variable:
my = NODAL model_coordinates(y) EXPRESSION variable: time = GLOBAL time
EVALUATE EXPRESSION = "(time/termTime)*(stretchx*(mx - 0.0) +
((my-0.25)/0.5)*stretchxy)" END
```

Assuming the above expression is being evaluated on nodes the current values for x and y model coordinates would be placed into mx and my and current analysis time placed into time

### 3.5.12. Expression Variable:

Scope: Definition For Function

---

Expression Variable: *Expr*

Parameter	Value	Default
<i>Expr</i>	string	undefined

**Summary** Specifies what the arguments of an expression exists, but does not define it correspond to. For example:

```
BEGIN DEFINITION FOR FUNCTION dx_shear TYPE = ANALYTIC
EXPRESSION variable: mx EXPRESSION variable: my EXPRESSION variable: time
EVALUATE EXPRESSION = "(time/termTime)*(stretchx*(mx - o.o) +
((my-o.25)/o.5)*stretchxy)" END
```

Call function must determine what each variable actually is based off of the string name

### 3.5.13. Ordinate

Scope: Definition For Function

---

Ordinate {=|are|is} *Name...*

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Specifies a string identifier for the dependent variable. Optionally specify a scale and/or offset value which transforms the ordinate values into `scaled_ordinate = scale * (ordinate + ordinate_offset)`.

### 3.5.14. Ordinate Offset

Scope: Definition For Function

---

Ordinate Offset {=|are|is} *Ordinate\_offset*

Parameter	Value	Default
<i>Ordinate_offset</i>	real	undefined

**Summary** Alias for Y OFFSET

### 3.5.15. Ordinate Scale

Scope: Definition For Function

---

Ordinate Scale {=|are|is} *Ordinate\_scale*

Parameter	Value	Default
<i>Ordinate_scale</i>	real	undefined

Summary    Alias for Y SCALE

### 3.5.16. Scale By

Scope: Definition For Function

---

Scale By *x*

Parameter	Value	Default
<i>x</i>	real	undefined

Summary    Specifies a scale factor to be applied.

### 3.5.17. Type

Scope: Definition For Function

---

Type {=|are|is} *Type*

Parameter	Value	Default
<i>Type</i>	{analytic constant multicolumn piecewise linear piecewise analytic  piecewise constant piecewise linear  piecewise multivariate xtable}	undefined

Summary    Specifies the type of function.

### 3.5.18. X Offset

Scope: Definition For Function

---

X Offset {=|are|is} *X\_offset*

Parameter	Value	Default
<i>X_offset</i>	real	undefined

Summary    Sets an offset for the x-axis

### 3.5.19. X Scale

Scope: Definition For Function

---

X Scale {=|are|is} *X\_scale*

Parameter	Value	Default
<i>X_scale</i>	real	undefined

---

**Summary** Sets a scale factor for the x-axis

### 3.5.20. Y Offset

Scope: Definition For Function

---

Y Offset {=|are|is} *Y\_offset*

Parameter	Value	Default
<i>Y_offset</i>	real	undefined

---

**Summary** Sets an offset for the y-axis

### 3.5.21. Y Scale

Scope: Definition For Function

---

Y Scale {=|are|is} *Y\_scale*

Parameter	Value	Default
<i>Y_scale</i>	real	undefined

---

**Summary** Sets a scale factor for the y-axis

## 3.6. VALUES

Scope: Definition For Function

---

Begin Values *empty*

*Xyvalues...*

End

---

**Summary** Lists the values of the function. The values should be listed one pair per line, independent variable first, with whitespace or comma as a separator.

### 3.6.1.

Scope: Values

---

*Xyvalues...*

Parameter	Value	Default
<i>Xyvalues</i>	real...	undefined

---

**Summary** For a piecewise linear function, lists an x-y pair for the nth interpolation point.

## 3.7. RESTART OVERVIEW

Sierra Framework services provide convenient utilities for restarting an analysis from previous results. The most general capability supplements the results of a previous analysis with internal state variables to continue an analysis. In this case the input mesh is supplied from the Input Database Name from the Finite Element Model command block 3.1 and the restart information is obtained from the the Input Database Name from the Restart Data command block. Continuation of a job using restart data output is invoked using the command line which follows.

### 3.7.1. Restart Time

Scope:

---

Restart Time {=|are|is} *Time*

Parameter	Value	Default
<i>Time</i>	real	undefined

---

**Summary** Specify restart file read at a specified time.

**Description** NOTE: This command must be placed at the Sierra scope of the input file.

Specify the time that the analysis will be restarted. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data. The restart 'time' must be greater than zero and less than or equal to the termination time.

By default, use of this command will cause previous output files (e.g., results, history, heartbeat, restart) to be overwritten. If this command is chosen, the onus is placed on the user to ensure that previous output files are not overwritten.

# 4. SOLUTION CONTROL REFERENCE

## 4.1. OVERVIEW

Arpeggio uses the *solution control* (SC) library from the SIERRA Framework to orchestrate execution of simulations. All Arpeggio input files must include a Solution Control Description block in the Procedure section of the input file. This description contains directives for executing either a steady-state (sequential) or transient analysis either of which can include nested nonlinear iteration or subcycling. Within the description one selects a named solution control system where the details of execution are more clearly spelled out. Because there are similarities between the Sequential, Transient, Nonlinear Iteration and Subcycling many operations are shared between these directives. However, each of these segments must be uniquely named internally so they can be properly managed under solution control.

Within each SC system, execution of a problem defined at the Region level corresponds to an Advance directive. Thus a steady-state analysis could conceivably be carried out with a single Advance directive. For transient analysis the system can contain several time blocks, each with a corresponding Advance directive. Examples of different control structures are given below.

As an example, the solution control command block for a steady-state Aria analysis would reflect the structure indicated below:

```
Begin Sierra myJob
.
. Materials, Solvers, Finite Element Model

Begin Procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential MySolveBlock
        Advance myRegion
      End
    End
  End
End

Begin Aria Region myRegion
.
```



```

        . ICs, BCs, equations, output instructions
        .
        . myRegion output
        .
    End Aria Region myRegion

End Procedure myProcedure
.
End Sierra myJob

```

A solution control command block for steady-state analysis containing nonlinear iteration for Aria and Adagio would reflect the general structure indicated below. Note that advancement of the solution can be governed by a user specified criteria, Parameters for Nonlinear Iteration:

```

Begin Sierra myJob
.
. Materials, Solvers, Finite Element Model

Begin Procedure myProcedure

    Begin Solution Control Description
        Use System Main
        Begin System Main
            Begin Sequential MySolveBlock
                Begin Nonlinear Iteration
                    Advance myAriaRegion
                    Advance myAdagioRegion
                    transfer adagio_to_aria
                End Nonlinear Iteration
            End
        End
    End

Begin transfer adagio_to_aria
.
. transfer commands
.
End transfer adagio_to_aria

Begin Aria Region myAriaRegion
.
. ICs, BCs, equations
. myAriaRegion output
.
End Aria Region myAriaRegion

```

```
Begin Adagio Region myAdagioRegion
.
. ICs, BCs, equations
. myAdagioRegion output
.
End Adagio Region myAdagioRegion

End Procedure myProcedure
.
End Sierra myJob
```

In the case of transient analysis the solution control command block will contain specification of times for which the analysis will be carried out. Additionally parameters defining the time integration must also be supplied by the user. Details concerning time integration parameters are included in the user manual for the application. A simple example the solution control command block for transient analysis would resemble the structure indicated below:

```

Begin Sierra myJob
.
. Materials, Solvers, Finite Element Model
.
Begin Procedure My_Aria_Procedure

Begin Solution Control Description

Use System Main

Begin System Main
Simulation Start Time           = 0.0
Simulation Termination Time     = 10.0
Simulation Max Global Iterations = 1000

Begin Transient Time_Block_1
Advance My_Aria_Region
End
Begin Transient Time_Block_2
Advance My_Aria_Region
End

End

Begin Parameters For Transient Time_Block_1
Start Time           = 0.0
Number of steps = 8
Begin Parameters For Aria Region My_Aria_Region
Time Step Variation   = Fixed
Initial Time Step Size = 0.001
End
End

Begin Parameters For Transient Time_Block_2
Begin Parameters For Aria Region My_Aria_Region
Time Step Variation   = Adaptive
Initial Time Step Size = 0.001
Predictor-Corrector Tolerance = 1e-3
Minimum Time Step Size   = 1e-6

```

```

        End
    End

    End

    .
End Procedure My_Aria_Procedure
.
End Sierra myJob

```

Similarly subcycled iterations in a one-way coupling between Aria and Presto could also be carried out in a transient analysis. In this case Presto subcycles at a small time, Aria has a larger time step and Aria is advanced when the two time steps arrive at the same solution time.

```

Begin Sierra myJob
.
Begin Procedure My_Aria_Procedure

    Begin Solution Control Description

        Use System Main

        Begin System Main
            Simulation Start Time          = 0.0
            Simulation Termination Time    = 10.0
            Simulation Max Global Iterations = 1000

            Begin Transient Time_Block_1
                Transfer Presto_to_Aria
                Advance My_Aria_Region
                Begin Subcycle PrestoSubcycle
                    Transfer Aria_to_Presto
                    Advance PrestoRegion
                End
            End

        End

    End

    Begin Parameters For Transient Time_Block_1
        Start Time          = 0.0
        Number of steps = 8

        Begin Parameters For Aria Region My_Aria_Region
            Time Step Variation    = Fixed
            Initial Time Step Size = 0.001
        End
    End
End

```

```

        Begin Parameters for Presto Region PrestoRegion
            initial time step = 1.0e-6
            # time step scale factor = 1.0
            time step increase factor = 10.
            # step interval = 500
        End
    End

End

.

Begin Aria Region myAriaRegion
.
.
End Aria Region myAriaRegion

Begin Presto Region myPrestoRegion
.
.
End Presto Region myPrestoRegion

End Procedure myProcedure
.
End Sierra myJob

```

It is important to note that Solution Control can orchestrate the execution of one Region or the execution of many Regions. Within a loosely-coupled code analysis SC is also used to control the movement of data between the coupled codes using the Transfer subsystem.

The outline views of various couplings include both Transfer and Advance events. In the examples above the event will always occur in the sequence specified. Alternatively one can specify that the event be carried out conditionally subject to criteria described syntactically as a "C" language *[When – expression]* where the expression criteria includes internal code variables or explicit evaluations. Here the input *[When – expression]* is parsed and transformed into an executable "C" statement. While some of the internal code variables used by a *[When – expression]* are intuitive (i.e. CURRENT\_TIME and CURRENT\_STEP) many others are application dependent. The most widely used explicit evaluations are measures of convergence based upon solution residuals `adagio.norm(0.0)` for solid mechanics applications and `aria.MaxResidualNorm(0.0)` for thermal-fluid applications. Several examples of *[When – expression]* are given below noting that the "C" expression must be enclosed in quotes within the input file.

Convergence based upon comparison of application residuals:

```

Begin parameters for nonlinear converge_step_p1

```

```
# following two lines shown must be a single input command line
converged when $(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0)
               < 1.e-6) || CURRENT_STEP > 2000"
End parameters for nonlinear converge_step_p1
```

Transfer at first step and then every four steps:

```
Transfer aria_to_adagio when "(CURRENT_STEP == 1) || (CURRENT_STEP % 4 == 0)"
```

Advance the region at second step:

```
advance aria_region when "CURRENT_STEP == 2"
```

Additionally, one may also use application specific global variables in the *[When – expression]* criteria. Global variables that are generally available for use are listed as such in the simulation log file. Unfortunately these variables may not be directly accessible to the user. Hence consultation with an application developer may be required in this regard.

In the case of transient analysis it is sometimes necessary to initialize a distribution of values before the analysis actually begins. As an example, one may want to initialize a Field that will be transferred to another Region with a distribution of values with the goal of setting a reference state. For this purpose solution control provides a means of initialization, Initialize.

```

Begin Sierra myJob
.
. Materials, Solvers, Finite Element Model
.
Begin Procedure My_Aria_Procedure

Begin Initialize
  Transfer var1_Region_to_var2_My_Aria_Region
End Initialize

Begin Solution Control Description

  Use system Initialize
  Use System Main

Begin System Main
  Simulation Start Time           = 0.0
  Simulation Termination Time     = 10.0
  Simulation Max Global Iterations = 1000

  Begin Transient Time_Block_1
    Advance My_Aria_Region
    Advance var1_Region
  End
End

Begin Parameters For Transient Time_Block_1
  Start Time           = 0.0
  Number of steps = 8
  Begin Parameters For Aria Region var1_Region
    . parameter commands
  End
  Begin Parameters For Aria Region My_Aria_Region
    . parameter commands
  End
End

End

.
. Var1_Region commands

```

```
.  
  . My_Aria_Region commands  
.  
End Procedure My_Aria_Procedure  
.  
End Sierra myJob
```



There are certain steps one will have to take when it is desired to just advance either the Aria or Adagio region in one of the transient blocks in an Arpeggio simulation.

The example shown below displays an Adagio-only second transient block. All transferred fields from the disabled app to the still-enabled app need to be handled using the steps described below. With this example, the only relevant transferred field is the temperature.

The first step taken was to switch new and old temperature states in Aria between the transient blocks. This will allow the Adagio region in the next transient block to use the latest temperature solution from Aria. The other notable syntax in the second transient block is to keep the aria to adagio transfer command in place. This input syntax will make sure the Adagio region uses a consistent temperature value while the region is being advanced.

```
Use System Main
Begin System Main
  Begin Transient time_block
    Advance aria_region

    #begin subcycle
    Transfer aria_to_adagio
    Advance adagio_region
    #end subcycle

    Transfer adagio_to_aria

  End Transient time_block

  Transfer T_switch

  Begin Transient time_block2
#    Advance aria_region

    #begin subcycle
    Transfer aria_to_adagio
    Advance adagio_region
    #end subcycle

  End Transient time_block2
End System Main

Begin Transfer aria_to_adagio
  Copy Volume Nodes from aria_region to adagio_region
  Send Field solution->TEMPERATURE State NEW to Temperature State NEW
End Transfer aria_to_adagio
```

```

Begin Transfer adagio_to_aria
  Copy Volume Nodes from adagio_region to aria_region
  Send Field DISPLACEMENT State New to Solution->Mesh_Displacements State New
End Transfer adagio_to_aria

Begin Transfer T_switch
  Copy Volume Nodes From Aria_region to Aria_region
  Send Field solution->Temperature State Old to solution->Temperature State New
  Send Field solution->Temperature State New to solution->Temperature State Old
End

```

## 4.2. SOLUTION CONTROL DESCRIPTION

Scope: Procedure

---

```

Begin Solution Control Description Name

  Use System Name

  Begin Initialize Name

  End

  Begin Parameters For

  End

  Begin System Name

  End

End

```

---

**Summary**    Contains the commands needed to execute an analysis using the arpeggio procedure that utilizes Solver Control.

### 4.2.1. Use System

Scope: Solution Control Description

---

Use System *Name*

Parameter <i>Name</i>	Value string	Default undefined
<hr/>		

**Summary**    This set the name of which system to use.

## 4.3. SYSTEM

Scope: Solution Control Description

---

Begin System *Name*

Adapt *Region\_name...* Using *Field\_name...* [ When *When-expression* ]

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Event *Name...* [ When *When-expression* ]

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Output *Name* [ When *When-expression* ]

Simulation Max Global Iterations {=|are|is} *Number*

Simulation Start Time {=|are|is} *Number*

Simulation Termination Time {=|are|is} *Number*

Transfer *Name* [ When *When-expression* ]

Use Initialize *Name*

Begin Adaptivity *Name*

End

Begin Sequential *Name*

End

Begin Transient *Name*

End

---

End

**Summary** This block wraps a solver system for a given name. The NAME parameter is the name used to define the system. There can be more than one system block in the Solver Control Description block. The "use system NAME" line command controls which one is to be used.

### 4.3.1. Adapt

Scope: System

---

Adapt *Region\_name...* Using *Field\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Field_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.3.2. Compute Indicator On

Scope: System

---

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Indicator_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.3.3. Event

Scope: System

---

Event *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

### 4.3.4. Execute Postprocessor Group

Scope: System

---

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Group_name</i>	string...	undefined
<i>Region_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to cause the group named *group\_name* to be executed on region *region\_name*.

### 4.3.5. Indicatemarkadapt

Scope: System

---

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Indicator</i>	string	undefined
<i>Marker</i>	string	undefined

---

**Summary** Shortcut line command... equivalent to: Compute Indicator On ... Mark ... Adapt ...

### 4.3.6. Mark

Scope: System

---

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Marker_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.3.7. Markadapt

Scope: System

---

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Marker</i>	string	undefined

---

Summary    Shortcut line command... equivalent to: Mark ... Adapt ...

### 4.3.8. Output

Scope: System

---

Output *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

Summary    A Solver Control Output line command which execute a perform I/O on the region.

### 4.3.9. Simulation Max Global Iterations

Scope: System

---

Simulation Max Global Iterations {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	integer	undefined

---

Summary    The Total number of Solves.

### 4.3.10. Simulation Start Time

Scope: System

---

Simulation Start Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

---

Summary    Simulation starting time. (by default 0.0)

### 4.3.11. Simulation Termination Time

Scope: System

---

Simulation Termination Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

---

Summary    The drop dead time.

### 4.3.12. Transfer

Scope: System

---

Transfer *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

Summary    A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of 'name' will be executed.

### 4.3.13. Use Initialize

Scope: System

---

Use Initialize *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

---

Summary    This set the name of which initialization to use.

## 4.4. TRANSIENT

Scope: System

---

Begin Transient *Name*

Adapt *Region\_name...* Using *Field\_name...* [ When *When-expression* ]

Advance *Name...* [ When *When-expression* ]

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Event *Name...* [ When *When-expression* ]

```

Execute Postprocessor Group Group_name... On Region_name... [
When When-expression ]

Indicatemarkadapt Region_name Using Indicator Marker [ When
When-expression ]

Involve Name

Mark Region_name... Using Marker_name... [ When When-expression
]

Markadapt Region_name Using Marker [ When When-expression ]

Output Name [ When When-expression ]

Transfer Name [ When When-expression ]

Begin Adaptivity Name

End

Begin Nonlinear Name

End

Begin Subcycle Name

End

End

```

---

**Summary** This block is used to wrap a time loop.

#### 4.4.1. Adapt

Scope: Transient

---

```

Adapt Region_name... Using Field_name... [ When When-expression ]

```

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Field_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.



### 4.4.2. Advance

Scope: Transient

---

Advance *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

### 4.4.3. Compute Indicator On

Scope: Transient

---

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Indicator_name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.4.4. Event

Scope: Transient

---

Event *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

## 4.4.5. Execute Postprocessor Group

Scope: Transient

---

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Group_name</i>	string...	undefined
<i>Region_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to cause the group named *group\_name* to be executed on region *region\_name*.

## 4.4.6. Indicatemarkadapt

Scope: Transient

---

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Indicator</i>	string	undefined
<i>Marker</i>	string	undefined

---

**Summary** Shortcut line command... equivalent to: Compute Indicator On ... Mark ... Adapt ...

## 4.4.7. Involve

Scope: Transient

---

Involve *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

#### 4.4.8. Mark

Scope: Transient

---

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Marker_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

#### 4.4.9. Markadapt

Scope: Transient

---

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Marker</i>	string	undefined

---

**Summary** Shortcut line command... equivalent to: Mark ... Adapt ...

#### 4.4.10. Output

Scope: Transient

---

Output *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** A Solver Control Output line command which execute a perform I/O on the region.

#### 4.4.11. Transfer

Scope: Transient

---

Transfer *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of 'name' will be executed.

## 4.5. NONLINEAR

Scope: Sequential

---

```
Begin Nonlinear Name

    Adapt Region_name... Using Field_name... [ When When-expression
    ]

    Advance Name... [ When When-expression ]

    Compute Indicator On Region_name... Using Indicator_name... [
    When When-expression ]

    Event Name... [ When When-expression ]

    Execute Postprocessor Group Group_name... On Region_name... [
    When When-expression ]

    Indicatemarkadapt Region_name Using Indicator Marker [ When
    When-expression ]

    Involve Name

    Mark Region_name... Using Marker_name... [ When When-expression
    ]

    Markadapt Region_name Using Marker [ When When-expression ]

    Output Name [ When When-expression ]

    Transfer Name [ When When-expression ]

    Begin Subcycle Name

    End

End
```

---

**Summary** This block is used to wrap a nonlinear solve loop.

### 4.5.1. Adapt

Scope: Nonlinear

---

```
Adapt Region_name... Using Field_name... [ When When-expression ]
```

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Field_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptmt on the specific block should be performed.

## 4.5.2. Advance

Scope: Nonlinear

---

Advance *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

## 4.5.3. Compute Indicator On

Scope: Nonlinear

---

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Indicator_name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

## 4.5.4. Event

Scope: Nonlinear

---

Event *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

### 4.5.5. Execute Postprocessor Group

Scope: Nonlinear

---

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Group_name</i>	string...	undefined
<i>Region_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to cause the group named *group\_name* to be executed on region *region\_name*.

### 4.5.6. Indicatemarkadapt

Scope: Nonlinear

---

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Indicator</i>	string	undefined
<i>Marker</i>	string	undefined

---

**Summary** Shortcut line command... equivalent to: Compute Indicator On ... Mark ... Adapt ...

### 4.5.7. Involve

Scope: Nonlinear

---

Involve *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

### 4.5.8. Mark

Scope: Nonlinear

---

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Marker_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.5.9. Markadapt

Scope: Nonlinear

---

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Marker</i>	string	undefined

---

**Summary** Shortcut line command... equivalent to: Mark ... Adapt ...

### 4.5.10. Output

Scope: Nonlinear

---

Output *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** A Solver Control Output line command which execute a perform I/O on the region.

### 4.5.11. Transfer

Scope: Nonlinear

---

Transfer *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of 'name' will be executed.

## 4.6. SUBCYCLE

Scope: Nonlinear

---

Begin Subcycle *Name*

Adapt *Region\_name...* Using *Field\_name...* [ When *When-expression* ]

Advance *Name...* [ When *When-expression* ]

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Event *Name...* [ When *When-expression* ]

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Involve *Name*

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Output *Name* [ When *When-expression* ]

Transfer *Name* [ When *When-expression* ]

End

---

**Summary** This block is used to wrap a subcycle time loop.

### 4.6.1. Adapt

Scope: Subcycle

---

Adapt *Region\_name...* Using *Field\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Field_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.



## 4.6.2. Advance

Scope: Subcycle

---

Advance *Name...* [ When *When-expression* ]

---

Parameter	Value	Default
<i>Name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

## 4.6.3. Compute Indicator On

Scope: Subcycle

---

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

---

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Indicator_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

## 4.6.4. Event

Scope: Subcycle

---

Event *Name...* [ When *When-expression* ]

---

Parameter	Value	Default
<i>Name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

## 4.6.5. Execute Postprocessor Group

Scope: Subcycle

---

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Group_name</i>	string...	undefined
<i>Region_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to cause the group named *group\_name* to be executed on region *region\_name*.

## 4.6.6. Indicatemarkadapt

Scope: Subcycle

---

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Indicator</i>	string	undefined
<i>Marker</i>	string	undefined

---

**Summary** Shortcut line command... equivalent to: Compute Indicator On ... Mark ... Adapt ...

## 4.6.7. Involve

Scope: Subcycle

---

Involve *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

### 4.6.8. Mark

Scope: Subcycle

---

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Marker_name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.6.9. Markadapt

Scope: Subcycle

---

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Marker</i>	string	undefined

**Summary** Shortcut line command... equivalent to: Mark ... Adapt ...

### 4.6.10. Output

Scope: Subcycle

---

Output *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

**Summary** A Solver Control Output line command which execute a perform I/O on the region.

### 4.6.11. Transfer

Scope: Subcycle

---

Transfer *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

**Summary** A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of 'name' will be executed.

## 4.7. SEQUENTIAL

Scope: System

---

Begin Sequential *Name*

Adapt *Region\_name...* Using *Field\_name...* [ When *When-expression* ]

Advance *Name...* [ When *When-expression* ]

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Event *Name...* [ When *When-expression* ]

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Involve *Name*

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Output *Name* [ When *When-expression* ]

Transfer *Name* [ When *When-expression* ]

Begin Adaptivity *Name*

End

Begin Nonlinear *Name*

End

End

---

**Summary** This block is used to wrap a sequential solution. It is used to wrap a sequence of Non-Linear or pseudo time solve step solves.

### 4.7.1. Adapt

Scope: Sequential

---

Adapt *Region\_name...* Using *Field\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Field_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.7.2. Advance

Scope: Sequential

---

Advance *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

### 4.7.3. Compute Indicator On

Scope: Sequential

---

Compute Indicator On *Region\_name...* Using *Indicator\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Indicator_name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

#### 4.7.4. Event

Scope: Sequential

---

Event *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

#### 4.7.5. Execute Postprocessor Group

Scope: Sequential

---

Execute Postprocessor Group *Group\_name...* On *Region\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Group_name</i>	string...	undefined
<i>Region_name</i>	string...	undefined

**Summary** Used within a Solver Control block to cause the group named *group\_name* to be executed on region *region\_name*.

#### 4.7.6. Indicatemarkadapt

Scope: Sequential

---

Indicatemarkadapt *Region\_name* Using *Indicator Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Indicator</i>	string	undefined
<i>Marker</i>	string	undefined

**Summary** Shortcut line command... equivalent to: Compute Indicator On ... Mark ... Adapt ...

#### 4.7.7. Involve

Scope: Sequential

---

Involve *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

**Summary** Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

### 4.7.8. Mark

Scope: Sequential

---

Mark *Region\_name...* Using *Marker\_name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string...	undefined
<i>Marker_name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a mesh adaptment on the specific block should be performed.

### 4.7.9. Markadapt

Scope: Sequential

---

Markadapt *Region\_name* Using *Marker* [ When *When-expression* ]

Parameter	Value	Default
<i>Region_name</i>	string	undefined
<i>Marker</i>	string	undefined

**Summary** Shortcut line command... equivalent to: Mark ... Adapt ...

### 4.7.10. Output

Scope: Sequential

---

Output *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

**Summary** A Solver Control Output line command which execute a perform I/O on the region.

### 4.7.11. Transfer

Scope: Sequential

---

Transfer *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

**Summary** A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of 'name' will be executed.

## 4.8. INITIALIZE

Scope: Solution Control Description

---

Begin Initialize *Name*

Advance *Name*... [ When *When-expression* ]

Event *Name*... [ When *When-expression* ]

Involve *Name*

Transfer *Name* [ When *When-expression* ]

End

---

**Summary** This block wraps a initializer for a given name. The NAME parameter is the name used to define the initialization block. There can be more than one initialize block in the Solver Control Description block. The "use initialize NAME" line command controls which one is to be used.

### 4.8.1. Advance

Scope: Initialize

---

Advance *Name*... [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

### 4.8.2. Event

Scope: Initialize

---

Event *Name*... [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

---

**Summary** Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.



### 4.8.3. Involve

Scope: Initialize

---

Involve *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

### 4.8.4. Transfer

Scope: Initialize

---

Transfer *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of 'name' will be executed.

## 4.9. PARAMETERS FOR

Scope: Solution Control Description

---

Begin Parameters For

Converged When *Convergence-expression*

Incremental Number Of Steps {=|are|is} *Number*

Initial Deltat {=|are|is} *Number*

Number Of Adaptivity Steps {=|are|is} *Number*

Number Of Steps {=|are|is} *Number*

Reinitialize Transient

Start Time {=|are|is} *Number*

Termination Time {=|are|is} *Number*

Time Step Quantum {=|are|is} *TimeStepQuantum*

Time Step Style *TimeStepStyle...*

Total Change In Time {=|are|is} *Number*

Begin Parameters For Aria Region *RegionName*

End

End

---

**Summary** A Solver Control PARAMETERS block to set up control data for the SC\_type parameter. Inside this block one sets the time step parameters or nonlinear parameters.

### 4.9.1. Converged When

Scope: Parameters For

---

Converged When *Convergence-expression*

Parameter	Value	Default
<i>Convergence-expression</i>	(expression)	undefined

**Summary** Set the convergence expression.

### 4.9.2. Incremental Number Of Steps

Scope: Parameters For

---

Incremental Number Of Steps {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	integer	undefined

**Summary** The incremental number steps to run the time for nonlinear loop. Number of time steps to run after restarting. NUMBER OF STEPS is total number of steps to run

### 4.9.3. Initial Deltat

Scope: Parameters For

---

Initial Deltat {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

**Summary** Assign an initial delta T

#### 4.9.4. Number Of Adaptivity Steps

Scope: Parameters For

---

Number Of Adaptivity Steps {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	integer	undefined

Summary The number steps to run the time or nonlinear loop

#### 4.9.5. Number Of Steps

Scope: Parameters For

---

Number Of Steps {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	integer	undefined

Summary The number steps to run the time for nonlinear loop

#### 4.9.6. Reinitialize Transient

Scope: Parameters For

Summary Reset time and re-initialize regions each step of the adaptivity loop.

#### 4.9.7. Start Time

Scope: Parameters For

---

Start Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

Summary Assign a start time.

#### 4.9.8. Termination Time

Scope: Parameters For

---

Termination Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

Summary Assign a final time to stop

### 4.9.9. Time Step Quantum

Scope: Parameters For

---

Time Step Quantum {=|are|is} *TimeStepQuantum*

Parameter	Value	Default
<i>TimeStepQuantum</i>	real	undefined

---

**Summary** Set the time stepping quantum time for SNAP style stepping.

### 4.9.10. Time Step Style

Scope: Parameters For

---

Time Step Style *TimeStepStyle...*

Parameter	Value	Default
<i>TimeStepStyle</i>	{clip noclip nosnap snap}	CLIP NOSNAP

---

**Summary** Set the time stepping style.

When CLIP is specified, the time step size will be clipped at the last step of the transient loop so that it ends at the transient loop's end time. If clip is not specified, the last time is allowed to exceed to the transient loop's end time and the following transient loop will start at the exceeded end time.

When SNAP is specified, the time step is broken down into "quantum" time units. By default this quantum time is 12 orders of magnitude down from the difference between the start and end time for the transient loop. This value can be overridden using the TIME STEP QUANTUM line command. All time values are "snapped" to multiples of the quantum time by rounding to the nearest quantum multiple.

### 4.9.11. Total Change In Time

Scope: Parameters For

---

Total Change In Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

---

**Summary** Use this number and the initial time to compute termination time.

### 4.9.12. Advance

Scope:

---

Advance *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

### 4.9.13. Converged When

Scope:

---

Converged When *Convergence-expression*

Parameter	Value	Default
<i>Convergence-expression</i>	(expression)	undefined

**Summary** Set the convergence expression.

### 4.9.14. Event

Scope:

---

Event *Name...* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string...	undefined

**Summary** Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

### 4.9.15. Initial Deltat

Scope:

---

Initial Deltat {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

**Summary** Assign an initial delta T

#### 4.9.16. Involve

Scope:

---

Involve *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

**Summary** Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

#### 4.9.17. Number Of Adaptivity Steps

Scope:

---

Number Of Adaptivity Steps {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	integer	undefined

**Summary** The number steps to run the time or nonlinear loop

#### 4.9.18. Number Of Steps

Scope:

---

Number Of Steps {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	integer	undefined

**Summary** The number steps to run the time for nonlinear loop

#### 4.9.19. Output

Scope:

---

Output *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

**Summary** A Solver Control Output line command which execute a perform I/O on the region.

#### 4.9.20. Reinitialize Transient

Scope:

**Summary** Reset time and re-initialize regions each step of the adaptivity loop.

### 4.9.21. Simulation Max Global Iterations

Scope:

---

Simulation Max Global Iterations {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	integer	undefined

Summary    The Total number of Solves.

### 4.9.22. Simulation Start Time

Scope:

---

Simulation Start Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

Summary    Simulation starting time. (by default 0.0)

### 4.9.23. Simulation Termination Time

Scope:

---

Simulation Termination Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

Summary    The drop dead time.

### 4.9.24. Start Time

Scope:

---

Start Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

Summary    Assign a start time.

## 4.9.25. Termination Time

Scope:

---

Termination Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

---

**Summary**    Assign a final time to stop

## 4.9.26. Time Step Quantum

Scope:

---

Time Step Quantum {=|are|is} *TimeStepQuantum*

Parameter	Value	Default
<i>TimeStepQuantum</i>	real	undefined

---

**Summary**    Set the time stepping quantum time for SNAP style stepping.

## 4.9.27. Time Step Style

Scope:

---

Time Step Style *TimeStepStyle...*

Parameter	Value	Default
<i>TimeStepStyle</i>	{clip noclip nosnap snap}	CLIP NOSNAP

---

**Summary**    Set the time stepping style.

When CLIP is specified, the time step size will be clipped at the last step of the transient loop so that it ends at the transient loop's end time. If clip is not specified, the last time is allowed to exceed to the transient loop's end time and the following transient loop will start at the exceeded end time.

When SNAP is specified, the time step is broken down into "quantum" time units. By default this quantum time is 12 orders of magnitude down from the difference between the start and end time for the transient loop. This value can be overridden using the TIME STEP QUANTUM line command. All time values are "snapped" to multiples of the quantum time by rounding to the nearest quantum multiple.



### 4.9.28. Total Change In Time

Scope:

---

Total Change In Time {=|are|is} *Number*

Parameter	Value	Default
<i>Number</i>	real	undefined

---

**Summary** Use this number and the initial time to compute termination time.

### 4.9.29. Transfer

Scope:

---

Transfer *Name* [ When *When-expression* ]

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of 'name' will be executed.

### 4.9.30. Use Initialize

Scope:

---

Use Initialize *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** This set the name of which initialization to use.

### 4.9.31. Use System

Scope:

---

Use System *Name*

Parameter	Value	Default
<i>Name</i>	string	undefined

---

**Summary** This set the name of which system to use.

# 5. TRANSFER REFERENCE

## 5.1. OVERVIEW

Recall that Sierra Mechanics supports application data associated with nodes, elements, faces or edges of a meshed discretization as in Figure 1.3-1. The Sierra Transfer utility provides the means by which to communicate data between two Sierra application Regions. Generally speaking the same type of data is most often communicated but data movement need not be for the same type, e.g. nodal data can be communicated to element data and vice-versa.

The Transfer utility is fairly flexible as it provides the ability to move data directly onto another problem domain either by direct copy or by interpolation. Analysts without prior experience with transfer are often uncertain as to which type of transfer to use. The two capabilities function exactly as their names imply but understanding which method to use requires a basic understanding of how each method works.

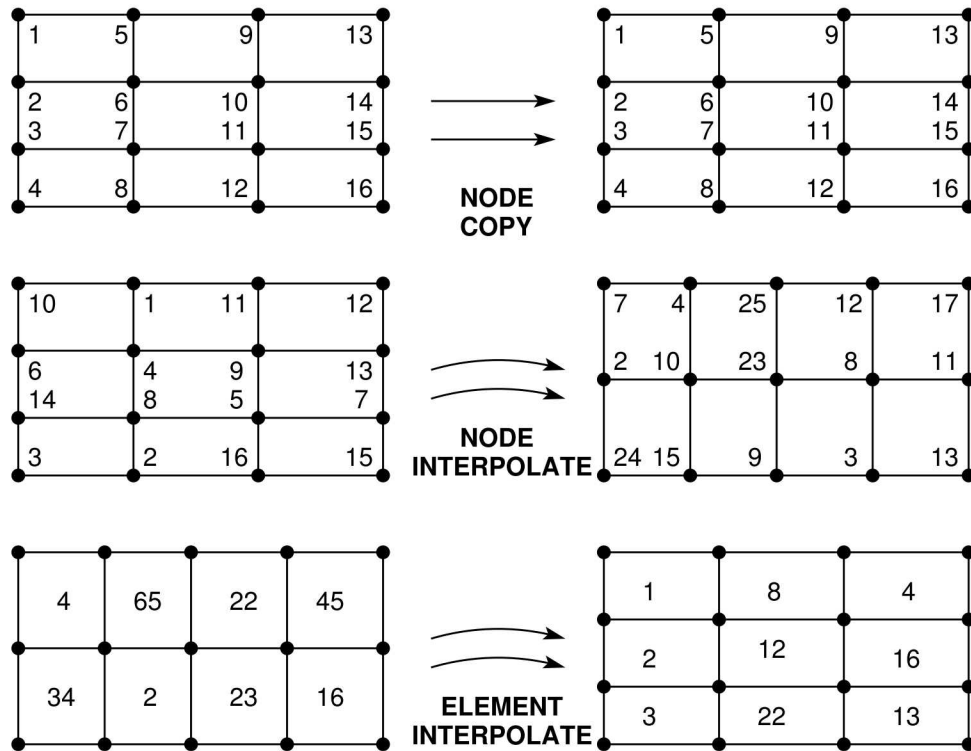
Copy transfer assumes that the discretization for applications involved in the transfer are identical. Moreover, copy transfer also assumes that the mesh is identical so that global IDs of nodes and elements within each mesh are the same. Under these assumptions a geometric search of source to destination locations is not necessary and a simple algorithm is able to perform the data transfer in a straightforward manner.

Interpolation transfer is much more general than copy transfer since it assumes only that data from one application must be geometrically mapped for use in another application. A mathematical definition of this mapping is made possible using the results from a geometric search of points on the destination mesh and their image on the sending mesh. With regard to code performance copy transfer will always more efficient than interpolation transfer but is rarely applicable in mainstream simulations. Interpolation transfer is designed to deal with complications that arise in mapping data from one application to the other and is more reliable. As a rule, one should always use interpolation transfer and not copy transfer. At the same time an analyst should strategize model construction so as to offset some of the performance costs of interpolation transfer.

Even with a basic understanding of transfer users of what transfer operations should be defined. Several proper transfer source and destinations are illustrated in Figure 5.1-1, here the numbers on the figures correspond to the ExodusII global IDs of nodes or elements.

Problematic transfer source and destination configurations are illustrated in Figure 5.1-2. Once again the numbers on the figures correspond to the ExodusII global IDs of nodes or elements.

In using the transfer utility one must clearly define the sending region (where the data resides) and the the receiving region (the data destination). Additionally one must also specify the general geometric



**Figure 5.1-1.. Valid Transfer Operations**

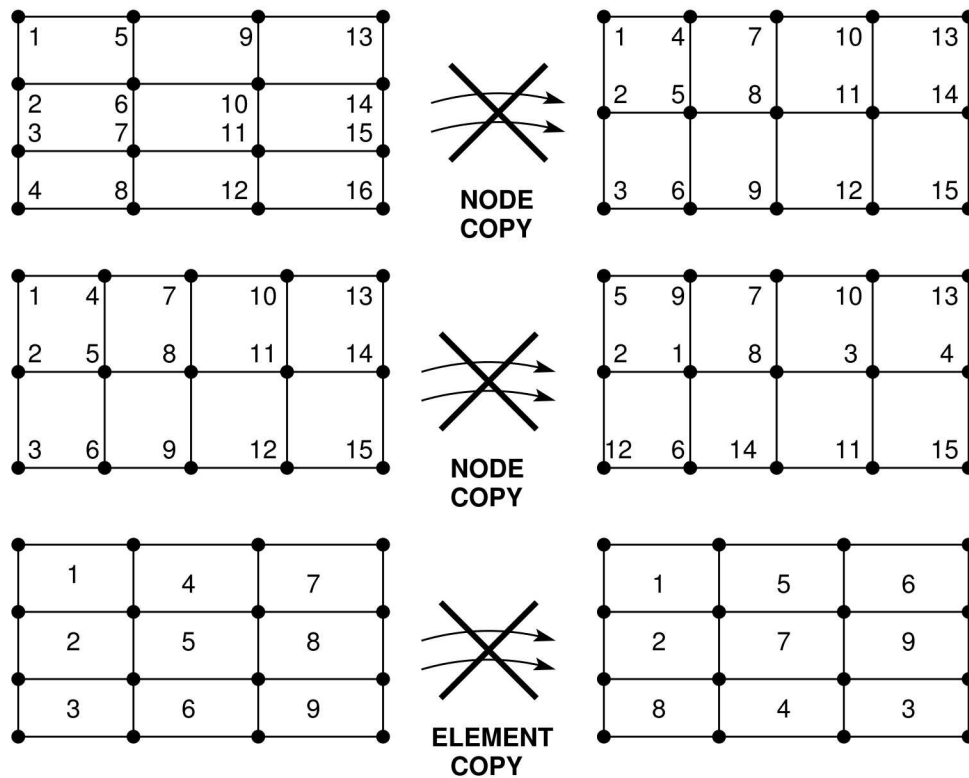
location of data sender and receiver based upon existing mesh entities (blocks or surfaces). Sender and receiver need not be of same topology but the source and target destinations should overlap geometrically. Clearly the definition mesh entities influences time spent in the geometric search process and should be a key consideration in model construction.

The following section outlines the commands to be used in setting up transfer operations. Special attention should be paid to the syntax of the SEND command line since it differs between COPY and INTERPOLATION transfer.

Since several different uses of transfer can arise and several of those examples for steady problems are included below. The same basic setup of transfer would apply to transient problems as well.

A skeleton outline of one-way transfer from Region\_1 to Region\_2 in a steady-state problem would be:

```
Begin Sierra
.
Begin Transfer my_transfer
.
transfer commands for first_region to second_region
.
End
.
```



**Figure 5.1-2.. Invalid Transfer Operation**

```
Begin Procedure My_Aria_Procedure
```

```
.
Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Sequential MySolveBlock
      Advance first_Region
      transfer my_transfer
      Advance second_Region
    End
  End
End
```

```
Begin Aria Region first_region
```

```
.
  eq energy for temperature On block_1 using q1 with lumped_mass diff
.
End
```

```
Begin Aria Region second_region
```

```
.
```

```
        eq energy for temperature On block_1 using q1 with xfer
    .
End

End

.
End Sierra
```

A skeleton outline of two-way transfer between Region\_1 to Region\_2 in a steady-state problem would be:

```
Begin Sierra
```

```
.
Begin Transfer my_first_transfer
.
  transfer commands for first_region to second_region
.
End
```

```
.
Begin Transfer my_second_transfer
.
  transfer commands for second_region to first_region
.
End
```

```
.
Begin Procedure My_Aria_Procedure
.
  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential MySolveBlock
        Advance first_Region
        transfer my_first_transfer
        Advance second_Region
        transfer my_second_transfer
      End
    End
  End
End
```

```
Begin Aria Region first_region
.
  eq energy for temperature On block_1 using q1 with diff
  eq species_3 for temperature On block_1 using q1 with xfer
.
End
```

```
Begin Aria Region second_region
.
  eq energy for temperature On block_1 using q1 with xfer
  eq species_3 for species_3 On block_1 using q1 with diff
.
End
```

End

.

End Sierra

Assume an input mesh for an Input\_Output Region [6.1](#) contains a nodal variable ConvCoeff. In this case a skeleton outline for one-way transfer of ConvCoeff to to Region\_2 in a steady-state problem would be:

```

Begin Sierra
.
Begin Transfer my_first_transfer
.
    transfer commands for input_output_region to second_region
.
    SEND field hNd state none TO ConvCoeff state none
.
End
.
Begin Procedure My_Aria_Procedure
.
    Begin Solution Control Description
        Use System Main
        Begin System Main
            Begin Sequential MySolveBlock
                Advance first_Region
                transfer my_first_transfer
                Advance second_Region
            End
        End
    End
End

Begin Input_Output io_region
    USE FINITE ELEMENT MODEL my_input_transfer
End

Begin Aria Region second_region
.
    USER FIELD REAL NODE SCALAR ConvCoeff on surface_1
.
End

End
.
End Sierra

```

## 5.2. TRANSFER

Scope: Procedure



---

Begin Transfer *Transfer\_name*

Abort If Field Not Defined On Copy Transfer Send Or Receive Object

All Fields

Copy *Option1 Option2* From *From\_region\_name* To *To\_region\_name*

Distance Function Is Closest Receive Node To Send Centroid

Exclude Ghosted

From *Option1* To *Option2*

Gauss Point Integration Order {=|are|is} *Order*

Interpolate *Option1 Option2* From *From\_region\_name* To  
*To\_region\_name*

Interpolation Function *User\_Subroutine*

Nodes Outside Region {=|are|is} *Option*

Search Coordinate Field *Source\_field\_name* State *Option1* To  
*Destination\_field\_name* State *Option2*

Search Geometric Tolerance {=|are|is} *Geometric\_tolerance*

Search Surface Gap Tolerance {=|are|is} *Surface\_gap\_tolerance* [ Or  
Less ]

Search Type {=|are|is} [ *Option1 Option2 Option3* ]

Select One Receiver For Each Send Object

Select One Unique Receiver For Each Send Object

Send *Predefined-transfer* Fields

Send Block *From\_blocks...* To *To\_blocks...*

Send Field *Source\_field\_name* State *Option1* To  
*Destination\_field\_name* State *Option2* [ Lower Bound *Lower\_bound*  
Upper Bound *Upper\_bound* ]

Begin Receive Blocks

End

Begin Send Blocks

End

---

End

**Summary** transfer region/mesh information. the mechanics/variables information will get sorted out by the calling procedure.

### 5.2.1. Abort If Field Not Defined On Copy Transfer Send Or Receive Object

Scope: Transfer

**Summary** For testing purposes only. Normally mesh objects in the send or receive mesh which do not have the specified field defined on them are just ignored. This line command allows the construction of tests in which it is known that every mesh object should have the specified field defined on it and to abort if that field is not found.

### 5.2.2. All Fields

Scope: Transfer

**Summary** Select all fields for transfer that have same name and state for source and destination regions.

### 5.2.3. Copy

Scope: Transfer

---

Copy *Option1* *Option2* From *From\_region\_name* To *To\_region\_name*

Parameter	Value	Default
<i>Option1</i>	{surface   volume}	undefined
<i>Option2</i>	{constraints   elements   nodes}	undefined
<i>From_region_name</i>	string	undefined
<i>To_region_name</i>	string	undefined

---

**Summary** Copy transfer elements, nodes or constraints from one region to another. The copy transfer is very specific in that the sending and receiving mesh parts must have identical global ids for every element to be copied. The copy transfer works by iterating over all the mesh objects in the receiving mesh and using the global id of the receiving mesh object to find a mesh object in the sending mesh with the same global id. The field to transfer is then copied from the sending to receiving objects. There is no interpolation and the actual coordinates of the sending and receiving objects are not used and could be very different. The copy transfer is used in very special cases where the same mesh was read into both the sending and receiving meshes, there was no element death and there was no adaptivity. In this special case, a copy transfer can be much faster than an interpolation transfer.

## 5.2.4. Distance Function Is Closest Receive Node To Send Centroid

Scope: Transfer

**Summary** To be used in conjunction with "SELECT ONE UNIQUE RECEIVER FOR EACH SEND OBJECT". This helped in the case where the sending and receiving element blocks did not overlap and an element transfer was using element centroids for the distance computation. The elements were very distorted so that a centroid of a surface element could be far from the surface. It was wanted that the receiving element be the one close to the surface of the block and close to the sending element in the adjacent block. Using the corner nodes was enough since it was a tet mesh with plane faces. In this particular and unusual case this alternative method of matching sending and receiving elements was useful, but it is not expected to be used often or maybe never again.

## 5.2.5. Exclude Ghosted

Scope: Transfer

**Summary** exclude ghosted nodes from a copy transfer

## 5.2.6. From

Scope: Transfer

---

From *Option1* To *Option2*

Parameter	Value	Default
<i>Option1</i>	{constraints   elements   nodes}	undefined
<i>Option2</i>	{constraints   elements   gauss_points   nodes}	undefined

**Summary** Allows the send/receive mesh objects to be different.

## 5.2.7. Gauss Point Integration Order

Scope: Transfer

---

Gauss Point Integration Order {= | are | is} *Order*

Parameter	Value	Default
<i>Order</i>	integer	undefined

**Summary** Integration order to use when transferring to Gauss points.

## 5.2.8. Interpolate

Scope: Transfer

---

Interpolate *Option1 Option2 From From\_region\_name To To\_region\_name*

Parameter	Value	Default
<i>Option1</i>	{surface   volume}	undefined
<i>Option2</i>	{constraints   elements   nodes}	undefined
<i>From_region_name</i>	string	undefined
<i>To_region_name</i>	string	undefined

---

**Summary** Interpolate will transfer elements, nodes or constraints from one mesh to another. The interpolation transfer is very general in that the field values to transfer will be interpolated from the sending to receiving mesh based on the coordinates of the sending and receiving mesh objects.

Many line commands can be used to modify the behavior of the interpolation transfer but the basic algorithm is straightforward. Every mesh object in the receiving mesh is converted into a point. For elements this is the average of the nodal coordinates. An element in the sending mesh containing this point is found. If the field to transfer is nodal, the element shape functions are used to interpolate the nodal field to the receiving point. If the field to transfer is elemental, a bi-linear least squares fit based upon neighboring elements is first performed and then used to define the interpolation of the element field at the receiving point.

## 5.2.9. Interpolation Function

Scope: Transfer

---

Interpolation Function *User\_Subroutine*

Parameter	Value	Default
<i>User_Subroutine</i>	string	undefined

---

**Summary** Allows an application defined subroutine to be used for the interpolation. Normally the interpolation transfer will determine the best type of interpolation to use: Basis functions for nodal fields and a neighborhood least squares fit for element fields. This line command can be used to override this if needed. It also allows an application to register it's own special interpolation functions that can then be used if the special name it was registered with is known.

## 5.2.10. Nodes Outside Region

Scope: Transfer

---

Nodes Outside Region {=|are|is} *Option*

Parameter <i>Option</i>	Value {extrapolate   ignore   project   truncate}	Default undefined
----------------------------	---	----------------------

---

**Summary** This line command defines what to do when a receiving point is outside the scope of the sending mesh.

IGNORE - The receiving mesh object can be ignored and will receive no value. This is almost never a good idea as it can cause mesh objects just outside to have a zero value when the nodes just inside the mesh might have very large values. This can result in a discontinuous receiving field.

EXTRAPOLATE - This is the default behavior. The sending field is extrapolated beyond the bounds of the sending mesh. This can lead to extrapolation error, such as when a large gradient at the surface causes a negative values when only positive values are acceptable. If this happens to the upper and lower bounds that can be placed on the fields to be transferred with the SEND FIELD command.

TRUNCATE - The receiving coordinate is projected back to the surface of the sending mesh to determine a value. This ensures that the receiving value is outside of the field values in the sending mesh.

PROJECT - This option is similar to TRUNCATE in which the receiving coordinate is projected back to the surface of the sending mesh to determine a value. In this case more effort is made to make sure that the projection is normal to the surface in the sending mesh. Sometimes gives a better result than Truncate but is a little more expensive to compute.

If the PROJECT option is used in transferring of surface values, the sending mesh should envelope the receiving mesh. Failure to satisfy this condition will generally result in failure of the transfer.

## 5.2.11. Search Coordinate Field

Scope: Transfer

---

Search Coordinate Field *Source\_field\_name* State *Option1* To  
*Destination\_field\_name* State *Option2*

Parameter	Value	Default
<i>Source_field_name</i>	string	undefined
<i>Option1</i>	{new   nm1   nm2   nm3   nm4   none   old}	undefined
<i>Destination_field_name</i>	string	undefined
<i>Option2</i>	{new   nm1   nm2   nm3   nm4   none   old}	undefined

**Summary** Normally the interpolation transfers use the default coordinate field to determine geometry information. This line command can be used to specify an alternate field.

## 5.2.12. Search Geometric Tolerance

Scope: Transfer

---

Search Geometric Tolerance {=|are|is} *Geometric\_tolerance*

Parameter	Value	Default
<i>Geometric_tolerance</i>	real	undefined

## 5.2.13. Search Surface Gap Tolerance

Scope: Transfer

---

Search Surface Gap Tolerance {=|are|is} *Surface\_gap\_tolerance* [ Or Less ]

Parameter	Value	Default
<i>Surface_gap_tolerance</i>	real	undefined

**Summary** This is a tricky parameter best ignored, let it default to some small number. During the interpolation transfer there is a geometric search based on the coordinates of the send and receive objects. As part of this search, an axis aligned bounding box is contracted for each sending object and SEARCH GAP TOLERANCE is used to make this box bigger than just a tight bounding box. Lists of receiving points are then quickly found within these axis aligned boxes.

If all points in the receiving mesh are within at least one box, no additional searching needs to be done and the search algorithm is fast. If there are still points in the receiving mesh that were outside of EVERY box, then a warning message will be issued about an "expensive search for extrapolation" for these points. This 'expensive search' can be very costly if a large number of receiving objects fall into this category and this line command is provided for those special cases.

The OR LESS optional parameter is used when the tolerance must be set to large value for one part of the mesh but much of the mesh needs a much smaller value. In some cases it is necessary for the tolerance to be set to the actual largest surface gap tolerance which may be far too large a gap for the rest of the mesh. Setting OR LESS allows the search tolerance to be reduced in areas of the mesh thus resulting in a faster search.

### 5.2.14. Search Type

Scope: Transfer

### 5.2.15. Select One Receiver For Each Send Object

Scope: Transfer

**Summary** This option will cause each sending object to be used once and only once. This will have the side effect of some receiving objects not getting any value at all. If you use this option, you will also want to set `NODES OUTSIDE REGION IGNORE`. The example which necessitated this option was a case in which there was a delta function defined on an element in the sending mesh. It was desirable that the delta functions be summed into the receiving mesh such that the total value of the sending was conserved. It was better to have only a single element on the receiving side have a non-zero value that was the sum of sending values and not worry about how close the receiving element was to the sending element. A check that this option is working is to use Encore to computer the sum of the values of the sending and receiving fields to make sure the total sum is the same.

### 5.2.16. Select One Unique Receiver For Each Send Object

Scope: Transfer

**Summary** An unusual flag to get around an odd problem. Normally each receive object transfers from the nearest sending object so it is almost always the case that a send object will be used multiple times to define a receiving value. This option will cause each sending object to be used only once. This will have the side effect of some receiving objects not getting any value at all. If you use this option, you will also want to set `NODES OUTSIDE REGION IGNORE` or else the uniqueness will be lost for nodes outside the sending region. The example which necessitated this option was a case in which there was a delta function defined on an element in the sending mesh. It was desirable that the delta function be defined on the receiving mesh for only a single element in the neighborhood of the sending element. The analysis was more sensitive to the number of delta functions on the receiving side than the location. So it was better to have only a single element on the receiving side have a non-zero value and not worry about how close the receiving element was to the sending element.

### 5.2.17. Send

Scope: Transfer

---

Send *Predefined-transfer* Fields

Parameter	Value	Default
<i>Predefined-transfer</i>	{}	undefined

**Summary** Use predefined transfer semantics provided by the specified name.

## 5.2.18. Send Block

Scope: Transfer

Send Block *From\_blocks...* To *To\_blocks...*

Parameter	Value	Default
<i>From_blocks</i>	string...	undefined
<i>To_blocks</i>	string...	undefined

**Summary** Add element blocks to a particular same mesh element copy transfer operator.

The copy transfer can have multiple of these lines to define many blocks, but each line sends a single block to a single block: SEND BLOCK block\_1 TO block\_1 SEND BLOCK block\_101 TO block\_101

The interpolation transfer can have only a single SEND BLOCK line, but can define many from/to blocks: SEND BLOCK block\_3 block\_5 block\_6 TO block\_3 block\_5

## 5.2.19. Send Field

Scope: Transfer

Send Field *Source\_field\_name* State *Option1* To *Destination\_field\_name* State *Option2* [ Lower Bound *Lower\_bound* Upper Bound *Upper\_bound* ]

Parameter	Value	Default
<i>Source_field_name</i>	string	undefined
<i>Option1</i>	{new   nm1   nm2   nm3   nm4   none   old}	undefined
<i>Destination_field_name</i>	string	undefined
<i>Option2</i>	{new   nm1   nm2   nm3   nm4   none   old}	undefined

**Summary** Specifies the mapping between source and destination field names. Vector and tensor fields can be subscripted using parenthesis and i's based or brackets and o based. Notes on subscripting: (0) Does not work for COPY transfers, only INTERPOLATION type transfers. (1) If the field name itself actually contains either parenthesis or brackets then we are in trouble and an error is going to be thrown due to a syntax error in index specification. (2) Only a single subscript is allowed so vectors of vectors or higher order tensors can not use double subscripts. But it should be possible to determine the correct offset within the field and pick out the correct value with a little effort. (3) Once subscripted, only a single value will be transferred. It is not possible to transfer multiple values starting at a certain index, instead multiple line commands must be used, as shown above. (4) The indexes can be o based with brackets or i based when using parenthesis.



Although this could be very confusing if mixed within a single line command. (5) Both the from and to fields can be subscripted independently on the same line.

example SEND FIELD velocity TO velocity SEND FIELD temp TO temperature lower bound 0 SEND FIELD x TO y lower bound 10 upper bound 100 SEND FIELD A(2) TO B(3) lower bound 10 upper bound 100 SEND FIELD A[1] TO B[2] lower bound 10 upper bound 100

# 6. INPUT OUTPUT REGION REFERENCE

## 6.1. INPUT\_OUTPUT REGION OVERVIEW

For some coupled simulations one can approximate part of the problem physics independent of the entire problem physics. In order to facilitate this type of loose application coupling the Sierra Framework provides the ability to input datasets that include the output of other simulations. An application can then make requests of information from these datasets. In fulfilling these requests, data can be extracted from these datasets and be copied or interpolated to another problem domain. Moreover these requests can be satisfied by data interpolated through time. The mechanism provided to achieve this end goal is known as the Input\_Output Region and its usage is described in what follows.

The input\_output region works in tandem with transfer [5.1](#) and solution control [4](#). Here transfer carries out the communication of data and solution control provides synchronization of the data transfer. Note that just like other Sierra Regions the input\_output region must have its own Finite Element model command block defined.

As an example, let us assume that an input mesh for an Input\_Output Region contains a nodal variable ConvCoeff that we wish to use in another Region. In this case an outline for one-way transfer of ConvCoeff to to a Region, *second\_region*, in a steady-state problem would be:

Begin Sierra

```
.
Begin Finite Element Model input_transfer
.
End
.
Begin Transfer my_first_transfer
.
transfer commands for input_output_region to second_region
.
SEND field hNd state none TO ConvCoeff state none
.
End
.
Begin Procedure My_Aria_Procedure
.
```

```

Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Sequential MySolveBlock
      Advance io_region
      transfer my_first_transfer
      Advance second_Region
    End
  End
End

Begin Input_Output io_region
  USE FINITE ELEMENT MODEL my_input_transfer
End

Begin Aria Region second_region
  .
  use Finite Element Model input_transfer
  .
  USER FIELD REAL NODE SCALAR ConvCoeff on surface_1
  .
End

End
.
End Sierra

```

## 6.2. INPUT\_OUTPUT REGION

Scope: Procedure

---

```

Begin Input_Output Region Parameter_block_name

  Create Element Field Field_name Of Type Option And Dimension
  Dimension [ Value {=|are|is} Number... ]

  Create Nodal Field Field_name Of Type Option And Dimension
  Dimension [ Value {=|are|is} Number... ]

  Fixed Time [ {=|are|is} Fixed_time ]

  Offset Time {=|are|is} Period_offset_time

  Periodicity Time {=|are|is} Periodicity_time

  Start Time {=|are|is} Start_time

```

Use Finite Element Model *ModelName* [ Model Coordinates Are  
*Nodal\_variable\_name* ]

Begin Heartbeat *Label*

End

Begin History Output *Label*

End

Begin Restart Data *Label*

End

Begin Results Output *Label*

End

End

---

**Summary** BEGIN INPUT TRANSFER model\_name USE FINITE ELEMENT MODEL fred  
 START TIME is 0 OFFSET TIME is 1 PERIODICITY TIME is 10 END INPUT  
 TRANSFER model\_name

### 6.2.1. Create Element Field

Scope: Input\_Output Region

---

Create Element Field *Field\_name* Of Type *Option* And Dimension *Dimension* [ *Value* {=|are|is} *Number...* ]

Parameter	Value	Default
<i>Field_name</i>	string	undefined
<i>Option</i>	{asym_tensor_03   complex   full_tensor_22   full_tensor_36   integer   long_integer   matrix_22   matrix_33   real   sym_tensor_21   sym_tensor_31   sym_tensor_33   unsigned_integer   unsigned_integer_64   vector_2d   vector_3d}	undefined
<i>Dimension</i>	integer	undefined

---

**Summary** Creates a Element Field name field\_name on the region.

## 6.2.2. Create Nodal Field

Scope: Input\_Output Region

---

Create Nodal Field *Field\_name* Of Type *Option* And Dimension *Dimension* [  
*Value* {=|are|is} *Number...* ]

Parameter	Value	Default
<i>Field_name</i>	string	undefined
<i>Option</i>	{asym_tensor_03   complex   full_tensor_22   full_tensor_36   integer   long_integer   matrix_22   matrix_33   real   sym_tensor_21   sym_tensor_31   sym_tensor_33   unsigned_integer   unsigned_integer_64   vector_2d   vector_3d}	undefined
<i>Dimension</i>	integer	undefined

---

**Summary** Creates a Nodal Field name *field\_name* on the region.

## 6.2.3. Fixed Time

Scope: Input\_Output Region

**Summary** The line specifies that the database will be read for a single, fixed time. Specifying the actual time is optional. If the time is not specified, the final time plane in the database will be read.

NOTE: This option take precedence over the periodic specifications given by START TIME, PERIODICITY TIME, and OFFSET TIME.

if FIXED TIME is specified then if FIXED TIME value is given then (eg., FIXED TIME is 1.) DATABASE TIME = FIXED TIME else (eg., FIXED TIME) DATABASE TIME = last time in database else if PERIODICITY TIME greater than 0 then if APPLICATION TIME less than or equal to START TIME then DATABASE TIME = APPLICATION TIME else DATABASE TIME = START TIME + (APPLICATION TIME - START TIME) modulo PERIODICITY TIME else DATABASE TIME = APPLICATION TIME now add OFFSET TIME to the computed DATABASE TIME

## 6.2.4. Offset Time

Scope: Input\_Output Region

---

Offset Time {=|are|is} *Period\_offset\_time*

Parameter	Value	Default
<i>Period_offset_time</i>	real	undefined

---

**Summary** This value is added to the application time to determine what database time slice to input. If OFFSET TIME were 15 than at application time 0 database time slice 15 would be read from the file and used for the initial values. At application time 1, database time slice 16 would be read. NOTE: The OFFSET TIME is added in after the START TIME and PERIODICITY TIME are used. The FIXED TIME option take precedence over this option.

if FIXED TIME is specified then if FIXED TIME value is given then (eg., FIXED TIME is 1.) DATABASE TIME = FIXED TIME else (eg., FIXED TIME) DATABASE TIME = last time in database else if PERIODICITY TIME greater than 0 then if APPLICATION TIME less than or equal to START TIME then DATABASE TIME = APPLICATION TIME else DATABASE TIME = START TIME + (APPLICATION TIME - START TIME) modulo PERIODICITY TIME else DATABASE TIME = APPLICATION TIME now add OFFSET TIME to the computed DATABASE TIME

## 6.2.5. Periodicity Time

Scope: Input\_Output Region

---

Periodicity Time {=|are|is} *Periodicity\_time*

Parameter	Value	Default
<i>Periodicity_time</i>	real	undefined

---

**Summary** START TIME and PERIODICITY TIME taken together give the time frame from the input database to use to initialize the application values. If START TIME is 25 and PERIODICITY TIME is 10, then time slices from 25 to 35 will be used over and over again as the application time runs from 0 to whatever. In general DATABASE TIME is (APPLICATION TIME - START TIME) modulo PERIODICITY TIME after the application time reaches the START TIME.

NOTE: The OFFSET TIME is added in after the START TIME and PERIODICITY TIME are used. The FIXED TIME option take precedence over this option.

if FIXED TIME is specified then if FIXED TIME value is given then (eg., FIXED TIME is 1.) DATABASE TIME = FIXED TIME else (eg., FIXED TIME) DATABASE TIME = last time in database else if PERIODICITY TIME greater than 0 then if APPLICATION TIME less than or equal to START TIME then DATABASE TIME = APPLICATION TIME else DATABASE TIME = START TIME + (APPLICATION TIME - START TIME) modulo PERIODICITY TIME else DATABASE TIME = APPLICATION TIME now add OFFSET TIME to the computed DATABASE TIME

### 6.2.6. Start Time

Scope: Input\_Output Region

---

Start Time {=|are|is} *Start\_time*

Parameter	Value	Default
<i>Start_time</i>	real	undefined

---

**Summary** The time in which to start applying PERIODICITY TIME. If PERIODICITY TIME is not specified then START TIME is ignored.

NOTES: The OFFSET TIME is added in after the START TIME and PERIODICITY TIME are used. The FIXED TIME option take precedence over this option.

if FIXED TIME is specified then if FIXED TIME value is given then (eg., FIXED TIME is 1.) DATABASE TIME = FIXED TIME else (eg., FIXED TIME) DATABASE TIME = last time in database else if PERIODICITY TIME greater than 0 then if APPLICATION TIME less than or equal to START TIME then DATABASE TIME = APPLICATION TIME else DATABASE TIME = START TIME + (APPLICATION TIME - START TIME) modulo PERIODICITY TIME else DATABASE TIME = APPLICATION TIME now add OFFSET TIME to the computed DATABASE TIME

### 6.2.7. Use Finite Element Model

Scope: Input\_Output Region

---

Use Finite Element Model *ModelName* [ Model Coordinates Are *Nodal\_variable\_name* ]

Parameter	Value	Default
<i>ModelName</i>	string	undefined

---

**Summary** Associates a predefined finite element model with this region.

## 7. EXAMPLES

Sierra application code couplings with Arpeggio can be carried out in a variety of ways. In this chapter a few simple problems are used to demonstrate some of the coupling approaches.

Here we note that success in performing the coupling hinges upon defining a proper setup for each of the application codes participating in the coupling. Understandably the coupling becomes more straightforward if one begins by first setting up each of the independent application code problems (i.e. an application **Region**) and later unites the **Regions** under Arpeggio.

The purpose of the examples is simply to demonstrate the basics of how the problem setup will differ for various use cases. The examples given here illustrate the use cases most likely to occur:

- One-way coupling of TF with Adagio from file on same mesh [7.1](#),
- One-way coupling of TF with Adagio from file on different mesh [7.2](#),
- One-way coupling of TF with Adagio on same mesh using transfer [7.3](#),
- Two-way coupling of TF with Adagio on same mesh [7.4](#),
- One way coupling of TF with another TF, same mesh [7.6](#),
- One way coupling of TF with Presto on same mesh with subcycling [7.5](#),

### 7.1. ONE-WAY COUPLING FROM FILE

In many problems of coupled physics one of the physics (primary) is dependent upon the other physics (secondary) but not vice-versa. In this case the coupling is considered to be one-way and can be accomplished simply by supplying a secondary physics solution to the primary physics simulation. In the context of problem solutions one would first solve the secondary physics problem and then communicate the solution to a primary physics simulation. Perhaps the easiest way to carry out such a simulation is to supply the secondary physics solution to the primary physics via file. The following example describes the process as it might be carried out in Arpeggio.



### 7.1.1. Problem Statement

Consider a one-way coupled thermal structural analysis problem in which a body is free to expand as a response to gradual temperature change in time. Although the problem geometry is changing due to the structural deformation, the geometry change is assumed to have minimal effect upon heat transfer in the body. For each time step, a heat conduction problem was solved for the temperature distribution using the Aria code and the results were written to file. The Aria output file is then used as the input file for Adagio where the temperatures are read into Adagio. Adagio subsequently solves for mechanical equilibrium which includes calculation of thermal strains due to changing temperatures.

Here we note that the thermal solution file time planes need not correspond to the Adagio time planes as the thermal solution will be interpolated in time to match the Adagio solution time. Furthermore, in this problem, an Aria results file is the Adagio input discretization so the problems correspond to the same mesh. Here it is important that the input Aria discretization contain the nodesets and sidesets needed to carry out the Adagio simulation. Problems in which one might wish to solve the Adagio problem on a different discretization can also be dealt with but in a slightly different manner.

### 7.1.2. Input File

```
begin sierra barOneWayCouple

  begin function analytic_sigma_zz
    type is analytic
    evaluate expression = "lambda=5.769231e5; mu = 3.84615e5; Delta = 25; alpha = 1e-4; -((3*lam
  end

  begin function THERMAL_STRAIN
    type is piecewise linear
    ordinate is strain
    abscissa is temperature
    begin values
      200.0    0.0
      400.0    0.02
    end values
  end function THERMAL_STRAIN

  define direction x with vector 1.0 0.0 0.0
  define direction y with vector 0.0 1.0 0.0
  define direction z with vector 0.0 0.0 1.0

  begin material linear_elastic
    density      = 0.1
    thermal strain function = THERMAL_STRAIN

    begin parameters for model elastic
      youngs modulus = 1.e6
```

```

    poissons ratio = 0.3
end parameters for model elastic

begin parameters for model elastic_plastic
    youngs modulus = 1.e6
    poissons ratio = 0.3
    yield stress = 1.0e6
    hardening modulus = 10.0
    beta is 0.999999
end parameters for model elastic_plastic

end material linear_elastic

begin finite element model mesh_arpeggio
    Database Name = 3dbar_temp.g
    Database Type = exodusII
    begin parameters for block block_1
        material linear_elastic
        model = elastic_plastic
    end parameters for block block_1
end finite element model mesh_arpeggio

begin procedure Arpeggio_Procedure

$=====
$  Add in solver control parameters
$=====

begin solution control description

    use system main

    begin system main

        begin transient mytransient
            advance adagio
        end transient mytransient

    end system main

begin parameters for transient mytransient
    start time = 0.0
    termination time = 2.0
    Number of Steps = 2
    begin parameters for adagio region adagio
        time increment = 1.0
    end
end

```

```

end

end solution control description

$=====
$ End of solver control parameters
$=====

# coupling is one_way using temperature distribution from file

$=====
$ Define the Adagio region
$=====

begin adagio region adagio

    use finite element model mesh_arpeggio

    begin user output
        include all blocks
        compute global analytic_sigma_zz as function analytic_sigma_zz
        compute global sigma_zz as max of element stress(zz)
        compute at every step
    end

    begin solution verification
        skip times = 0.0 to 1.0
        completion file = VerifSigmaZZ
        verify global sigma_zz = function analytic_sigma_zz
        tolerance = 1
    end

    begin prescribed temperature
        include all blocks
        read variable = temperature
    end

    ### definition of BCs ###
    begin fixed displacement
        surface = surface_10
        components = z
    end fixed displacement

    begin fixed displacement
        surface = surface_20
        components = z

```

```

end fixed displacement

### -----###
### Solver definition ###
### -----###

begin solver
  Begin cg
    Target relative Residual = 1.0e-11
    Maximum Iterations = 30
    Minimum Iterations = 1
    begin full tangent preconditioner
      automatic smoothing factor = 0.1
    end
  end
end

### output description ###
begin Results Output output_adagio
  Database Name = barOneWayCoupleFromFile_mech.e
  Database Type = exodusII
  At Step 0, Increment = 1
  nodal Variables = temperature as temperature
  nodal Variables = velocity as vel
  nodal Variables = displacement as displ
  element Variables = stress as stress
  global Variables = timestep as TIMESTEP
  global variables = external_energy as ExternalEnergy
  global variables = internal_energy as InternalEnergy
  global variables = kinetic_energy as KineticEnergy
  global variables = momentum as Momentum
end results output output_adagio

end adagio region adagio

end procedure Arpeggio_Procedure

end sierra barOneWayCouple

```

## 7.2. ONE-WAY COUPLING USING TRANSFER FROM DIFFERENT MESH

In some coupled physics one of the physics (primary) is dependent upon the other physics (secondary) but not vice-versa. In this case the coupling is considered to be one-way and can be accomplished simply by supplying a secondary physics solution to the primary physics simulation. In the context of problem

solutions one would first solve the secondary physics problem and then communicate the solution to a primary physics simulation. As previously demonstrated one way to carry out such a simulation is to supply the secondary physics solution to the primary physics via file 7.1. However, in some cases the secondary physics solution is available on a vastly different geometry. In this case the secondary physics solution must be interpolated onto the primary physics as needed. In Sierra Mechanics the communication step of such an analysis is carried out using Solution Control and Transfer operations. Here Transfer describes the information and Solution Control ensures sequencing of information to the primary physics. The following example describes the solution process to perform a coupled analysis using a precomputed thermal solution and Adagio.

### **7.2.1. Problem Statement**

Consider a one-way coupled thermal structural analysis problem in which a body is free to expand as a response to gradual temperature change in time. Although the problem geometry is changing due to the structural deformation, the geometry change is assumed to have minimal effect upon heat transfer in the body. For this situation a reasonable approach may be to precompute the heat transfer solution and then supply it to the mechanical simulation. Here a transient heat conduction problem on a full geometry was solved for the temperature distribution using the Aria code and the results were saved to file. Later on the previously computed temperature distribution was supplied to Adagio for solution of mechanical equilibrium which includes calculation of thermal strains due to changing temperatures. In this particular case the Adagio problem could be solved by invoking symmetry conditions so the model geometry is a subset of the thermal model geometry.

In this particular case the Adagio problem could be solved by invoking symmetry conditions so the model geometry is a subset of the thermal model geometry. During the simulation the transient thermal solution is read from file these results are then communicated to Adagio using a transfer operation. Once the Aria values are received by Adagio the structural problem is then solved. Since the thermal and structural model geometries are different, it is necessary to use the transfer INTERPOLATE operation. Note that the problem advances with the two applications lock stepped in time with the thermal solution is being interpolated in both space and time.

### **7.2.2. Input File**

## **7.3. ONE-WAY COUPLING USING TRANSFER**

In many problems of coupled physics one of the physics (primary) is dependent upon the other physics (secondary) but not vice-versa. In this case the coupling is considered to be one-way and can be accomplished simply by supplying a secondary physics solution to the primary physics simulation. In the context of problem solutions one would first solve the secondary physics problem and then communicate the solution to a primary physics simulation. One way to carry out such a simulation is to supply the secondary physics solution to the primary physics via file 7.1. However, in many instances it is more convenient to carry out both simulations simultaneously and directly communicate the secondary physics solution to the primary physics as needed. In Sierra Mechanics the communication step of such

an analysis is carried out using Solution Control and Transfer operations. Here Transfer describes the information and Solution Control ensures sequencing of information to the primary physics. The following example describes the solution process to perform a coupled analysis using Aria and Adagio.

### 7.3.1. Problem Statement

Consider a one-way coupled thermal structural analysis problem in which a body is free to expand as a response to gradual temperature change in time. Although the problem geometry is changing due to the structural deformation, the geometry change is assumed to have minimal effect upon heat transfer in the body. For each time step, a heat conduction problem was solved for the temperature distribution using the Aria code. Once the thermal solution has been obtained the temperature solution is communicated to Adagio via Transfer and Adagio then solves for mechanical equilibrium which includes calculation of thermal strains due to changing temperatures.

Note that the problem advances with the two applications lock stepped in time. In this problem the Aria input discretization is identical to that of Adagio. During the simulation an Aria solution is performed and Aria results are then communicated to Adagio using a transfer COPY operation. Once the Aria values are received by Adagio the structural problem is then solved. Problems in which one might wish to solve the Aria and Adagio problems on different discretizations can be dealt with by making simple modifications to the input replacing the transfer COPY operation with a INTERPOLATE operation.

### 7.3.2. Input File

```
begin sierra barOneWayCouple

  begin function analytic_sigma_zz
    type is analytic
    evaluate expression = "lambda=5.769231e5; mu = 3.84615e5; Delta = 25; alpha = 1e-4; -((3*lam
  end

  begin function THERMAL_STRAIN
    type is piecewise linear
    ordinate is strain
    abscissa is temperature
    begin values
      200.0    0.0
      400.0    0.02
    end values
  end function THERMAL_STRAIN

  define direction x with vector 1.0 0.0 0.0
  define direction y with vector 0.0 1.0 0.0
  define direction z with vector 0.0 0.0 1.0
```

```

begin material linear_elastic
  density      = 0.1
  thermal strain function = THERMAL_STRAIN

  begin parameters for model elastic
    youngs modulus  = 1.e6
    poissons ratio  = 0.3
  end parameters for model elastic

  begin parameters for model elastic_plastic
    youngs modulus  = 1.e6
    poissons ratio  = 0.3
    yield stress    = 1.0e6
    hardening modulus = 10.0
    beta is 0.999999
  end parameters for model elastic_plastic

end material linear_elastic

BEGIN AZTEC EQUATION SOLVER solve_temperature
  solution method      = cg
  preconditioning method = jacobi
  maximum iterations    = 1000
  residual norm scaling = r0
  residual norm tolerance = 1.0e-6
  debug output level    = 0
END AZTEC EQUATION SOLVER solve_temperature

begin ARIA MATERIAL linear_elastic
  thermal conductivity = constant k=401.0
  specific heat        = constant cp=385
  density              = constant rho=0.1
  heat conduction      = Fouriers_law
end ARIA MATERIAL linear_elastic

begin finite element model mesh_arpeggio
  Database Name = 3dbar.g
  Database Type = exodusII
  begin parameters for block block_1
    material linear_elastic
    model = elastic_plastic
  end parameters for block block_1
end finite element model mesh_arpeggio

begin procedure Arpeggio_Procedure

```

```

$=====
$  Add in solver control parameters
$=====

begin solution control description

    begin initialize mytransient_init
        advance AriaRegion
        transfer TariatoTadagio_init
        advance adagio
    end initialize mytransient_init

    use system main

    begin system main
        use initialize mytransient_init
        begin transient mytransient
            advance AriaRegion
            transfer TariatoTadagio
            advance adagio
        end transient mytransient

    end system main

    begin parameters for transient mytransient
        start time = 0.0
        termination time = 2.0
        Number of Steps = 2
        BEGIN PARAMETERS FOR ARIA REGION AriaRegion
            Initial Time Step Size = 1.0
            Time Step Variation = Fixed
        END

        begin parameters for adagio region adagio
            time increment = 1.0
        end
    end

end solution control description

$=====
$  End of solver control parameters
$=====

#coupling type is one_way

begin transfer TariatoTadagio_init

```



```

    copy volume nodes from AriaRegion to adagio
    send field solution->TEMPERATURE state new to temperature state old
    send field solution->TEMPERATURE state new to temperature state new
end transfer TariatoTadagio_init

```

```

begin transfer TariatoTadagio
    copy volume nodes from AriaRegion to adagio
    send field solution->TEMPERATURE state new to temperature state new
end transfer TariatoTadagio

```

```

$=====
$ Define the Aria region
$=====

```

```

BEGIN ARIA REGION AriaRegion

```

```

    use linear solver solve_temperature

```

```

    nonlinear solution strategy = newton
    maximum nonlinear iterations = 10
    nonlinear residual tolerance = 1.0e-6
    nonlinear relaxation factor = 1.0

```

```

    use finite element model mesh_arpeggio

```

```

    IC const          at block_1      Temperature = 273.0
    BC const dirichlet at nodelist_20 Temperature = 273.0
    BC const dirichlet at nodelist_10 Temperature = 373.0

```

```

    EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF

```

```

    ### output description ###

```

```

    BEGIN RESULTS OUTPUT LABEL diffusion output
        database Name = barOneWayCoupleTransfer_therm.e
        at step 0, increment = 1
        title Aria cube test
        nodal variables = solution->TEMPERATURE as Temperature
    END RESULTS OUTPUT LABEL diffusion output

```

```

END ARIA REGION AriaRegion

```

```

$=====
$ Define the Adagio region
$=====

```

```

begin adagio region adagio

```

```

use finite element model mesh_arpeggio

begin user output
  include all blocks
  compute global analytic_sigma_zz as function analytic_sigma_zz
  compute global sigma_zz as max of element stress(zz)
  compute at every step
end

begin solution verification
  skip times = 0.0 to 1.0
  completion file = VerifSigmaZZ
  verify global sigma_zz = function analytic_sigma_zz
  tolerance = 1
end

### definition of BCs ###
begin fixed displacement
  surface = surface_10
  components = z
end fixed displacement

begin fixed displacement
  surface = surface_20
  components = z
end fixed displacement

### -----###
### Solver definition ###
### -----###

begin solver
  Begin cg
    Target relative Residual = 1.0e-11
    Maximum Iterations = 30
    Minimum Iterations = 1
    begin full tangent preconditioner
      automatic smoothing factor = 0.1
    end
  end
end

end

### output description ###
begin Results Output output_adagio
  Database Name = barOneWayCoupleTransfer_mech.e
  Database Type = exodusII

```

```

    At Step 0, Increment = 1
    nodal Variables = temperature as temperature
    nodal Variables = velocity as vel
    nodal Variables = displacement as displ
    element Variables = stress as stress
    global Variables = timestep as TIMESTEP
    global variables = external_energy as ExternalEnergy
    global variables = internal_energy as InternalEnergy
    global variables = kinetic_energy as KineticEnergy
    global variables = momentum as Momentum
end results output output_adagio

end adagio region adagio

end procedure Arpeggio_Procedure

end sierra barOneWayCouple

```

## 7.4. TWO-WAY COUPLING WITH TRANSFER

### 7.4.1. Problem Statement

This is a test of solving a simple one-dimensional thermal diffusion problem with Dirichlet BCs. The test problem is shown schematically in Figure. Although the problem is one-dimensional we solve the problem in a three-dimensional setting. Once the diffusion problem has been solved numerically the temperature result is postprocessed to obtain a comparison with the analytical result and the distribution of diffusive heat flux. This test input also demonstrates the use tabular function and Encore function material property specification in Aria.

### 7.4.2. Input File

```

# Converted: gapClosure.i using c2a 0.12
begin sierra gapClosure

begin definition for function THERMAL_STRAIN
  type is piecewise linear
  ordinate = strain
  abscissa = temperature
  begin values
    373.0 0.0
    1373.0 12e-3
  end values
end definition for function THERMAL_STRAIN

```

```

begin definition for function tempLeft
  type is piecewise linear
  begin values
    0.0 373
    0.5 373
    1.0 773
    7.0 773
    7.5 373
    10.0 373
  end
end

begin definition for function tempRight
  type is piecewise linear
  begin values
    0.0 373
    1.0 373
    2.0 623
    3.0 873
    4.0 1123
    5.0 873
    6.0 623
    7.0 373
    10.0 373
  end
end

define direction x with vector 1.0 0.0 0.0
define direction y with vector 0.0 1.0 0.0
define direction z with vector 0.0 0.0 1.0

begin Aria material linear_elastic
  specific heat = constant  cp = 1.0
  density = constant  rho = 0.1
  heat conduction = basic
  thermal conductivity = constant  k = 250
end

begin property specification for material linear_elastic
  density = 0.1
  thermal strain function = THERMAL_STRAIN

  begin parameters for model elastic
    youngs modulus = 1.e7
    poissons ratio = 0.0
  end parameters for model elastic

```

```
end property specification for material linear_elastic
```

```
begin finite element model mesh_calagio
```

```
  Database Name = gapClosure.g
```

```
  begin parameters for block block_1
```

```
    material linear_elastic
```

```
    model = elastic
```

```
  end parameters for block block_1
```

```
  begin parameters for block block_2
```

```
    material linear_elastic
```

```
    model = elastic
```

```
  end parameters for block block_2
```

```
end finite element model mesh_calagio
```

```
begin aztec equation solver solve_temperature_aztec
```

```
  solution method = gmres
```

```
  preconditioning method = jacobi
```

```
  maximum iterations = 1000
```

```
  residual norm tolerance = 0.0000001
```

```
  residual norm scaling = NONE
```

```
end aztec equation solver solve_temperature_aztec
```

```
begin procedure Acca_Procedure
```

```
  begin solution control description
```

```
    Begin Initialize sys_init
```

```
      advance aria
```

```
      transfer aria_to_adagio_init
```

```
      advance adagio
```

```
    End initialize sys_init
```

```
    Use System Main
```

```
    Begin System Main
```

```
      use Initialize sys_init
```

```
      Simulation Start Time      = 0.0
```

```
      Simulation Termination Time = 8.0
```

```
      Begin Transient p1
```

```
        Begin Nonlinear converge_step_p1
```

```
          advance aria
```

```
          transfer aria_to_adagio
```

```
          advance adagio
```

```
          transfer adagio_to_aria
```

```
        End Nonlinear converge_step_p1
```

```

End Transient p1
Begin Transient p2
  Begin Nonlinear converge_step_p2
    advance aria
    transfer aria_to_adagio
    advance adagio
    transfer adagio_to_aria
  End Nonlinear converge_step_p2
End Transient p2
Begin Transient p3
  Begin Nonlinear converge_step_p3
    ### advance_aria_and_adagio
    advance aria
    transfer aria_to_adagio
    advance adagio
    transfer adagio_to_aria
  End Nonlinear converge_step_p3
End Transient p3
Begin Transient p4
  Begin Nonlinear converge_step_p4
    advance aria
    transfer aria_to_adagio
    advance adagio
    transfer adagio_to_aria
  End Nonlinear converge_step_p4
End Transient p4
Begin Transient p5
  Begin Nonlinear converge_step_p5
    advance aria
    transfer aria_to_adagio
    advance adagio
    transfer adagio_to_aria
  End Nonlinear converge_step_p5
End Transient p5
Begin Transient p6
  Begin Nonlinear converge_step_p6
    advance aria
    transfer aria_to_adagio
    advance adagio
    transfer adagio_to_aria
  End Nonlinear converge_step_p6
End Transient p6
Begin Transient p7
  Begin Nonlinear converge_step_p7
    advance aria
    transfer aria_to_adagio
    advance adagio

```

```

        transfer adagio_to_aria
    End Nonlinear converge_step_p7
End Transient p7
Begin Transient p8
    Begin Nonlinear converge_step_p8
        advance aria
        transfer aria_to_adagio
        advance adagio
        transfer adagio_to_aria
    End Nonlinear converge_step_p8
End Transient p8
End System Main

Begin parameters for nonlinear converge_step_p1
    converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
    # many other choices in documentation
End parameters for nonlinear converge_step_p1

Begin parameters for nonlinear converge_step_p2
    converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
    # many other choices in documentation
End parameters for nonlinear converge_step_p2

Begin parameters for nonlinear converge_step_p3
    converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
    # many other choices in documentation
End parameters for nonlinear converge_step_p3

Begin parameters for nonlinear converge_step_p4
    converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
    # many other choices in documentation
End parameters for nonlinear converge_step_p4

Begin parameters for nonlinear converge_step_p5
    converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
    # many other choices in documentation
End parameters for nonlinear converge_step_p5

Begin parameters for nonlinear converge_step_p6
    converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
    # many other choices in documentation
End parameters for nonlinear converge_step_p6

Begin parameters for nonlinear converge_step_p7
    converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
    # many other choices in documentation
End parameters for nonlinear converge_step_p7

```

```

Begin parameters for nonlinear converge_step_p8
  converged when "(aria.MaxResidualNorm(0.0) < 1.e-6 && adagio.norm(0.0) < 1.e-6) || CURRE
  # many other choices in documentation
End parameters for nonlinear converge_step_p8

begin parameters for transient p1
  start time = 0.0
  termination time = 1.0

  BEGIN PARAMETERS FOR Aria REGION Aria
    initial time step size = 0.5

    time step variation = fixed
    time integration method = first_order
  END PARAMETERS FOR Aria REGION Aria

  begin parameters for adagio region adagio
    time increment = 1.0
  end parameters for adagio region adagio

end

begin parameters for transient p2
  start time = 1.0
  termination time = 2.0

  BEGIN PARAMETERS FOR Aria REGION Aria
    initial time step size = 10.0

    time step variation = fixed
    time integration method = first_order
  END PARAMETERS FOR Aria REGION Aria

  begin parameters for adagio region adagio
    time increment = 10.0
  end parameters for adagio region adagio

end

begin parameters for transient p3
  start time = 2.0
  termination time = 3.0

  BEGIN PARAMETERS FOR Aria REGION Aria
    initial time step size = 1.0

```



```

        time step variation = fixed
        time integration method = first_order
END PARAMETERS FOR Aria REGION Aria

begin parameters for adagio region adagio
    time increment = 1.0
end parameters for adagio region adagio

end

begin parameters for transient p4
    start time = 3.0
    termination time = 4.0

    BEGIN PARAMETERS FOR Aria REGION Aria
        initial time step size = 0.5

        time step variation = fixed
        time integration method = first_order
    END PARAMETERS FOR Aria REGION Aria

    begin parameters for adagio region adagio
        time increment = 0.5
    end parameters for adagio region adagio

end

begin parameters for transient p5
    start time = 4.0
    termination time = 5.0

    BEGIN PARAMETERS FOR Aria REGION Aria
        initial time step size = 1.0

        time step variation = fixed
        time integration method = first_order
    END PARAMETERS FOR Aria REGION Aria

    begin parameters for adagio region adagio
        time increment = 1.0
    end parameters for adagio region adagio

end

begin parameters for transient p6
    start time = 5.0
    termination time = 6.0

```

```

BEGIN PARAMETERS FOR Aria REGION Aria
  initial time step size = 1.0

  time step variation = fixed
  time integration method = first_order
END PARAMETERS FOR Aria REGION Aria

begin parameters for adagio region adagio
  time increment = 1.0
end parameters for adagio region adagio

end

begin parameters for transient p7
  start time = 6.0
  termination time = 7.0

  BEGIN PARAMETERS FOR Aria REGION Aria
    initial time step size = 1.0

    time step variation = fixed
    time integration method = first_order
  END PARAMETERS FOR Aria REGION Aria

  begin parameters for adagio region adagio
    time increment = 1.0
  end parameters for adagio region adagio

end

begin parameters for transient p8
  start time = 7.0
  termination time = 8.0

  BEGIN PARAMETERS FOR Aria REGION Aria
    initial time step size = 0.5

    time step variation = fixed
    time integration method = first_order
  END PARAMETERS FOR Aria REGION Aria

  begin parameters for adagio region adagio
    time increment = 0.5
  end parameters for adagio region adagio

end

```

```

end solution control description

# default to: standard_aria_to_adagio_init
begin transfer aria_to_adagio_init
  copy volume nodes from aria to adagio
  send field solution->temperature state old to temperature state old
  send field solution->temperature state new to temperature state new
end

# default to: standard_aria_to_adagio_advance
begin transfer aria_to_adagio
  copy volume nodes from aria to adagio
  send field solution->temperature state new to temperature state new
end

begin transfer adagio_to_aria
  copy volume nodes from adagio to aria
  send field displacement state new to solution->mesh_displacements state new
end

begin aria region Aria

  EQ energy for temperature on all_blocks using Q1 with diff mass # src
  EQ mesh for mesh_displacements on all_blocks using Q1 with xfer

  Begin Initial Condition myICBlock
    temperature = 373.0
    add volume block_1
    add volume block_2
    $can have many of the above and below extent specifiers
    $nodeset = nodelist_1
    $surface = surface_1
  End Initial Condition myICBlock

  begin temperature boundary condition surface_14
    add surface surface_14
    temperature time function = tempLeft
  end
  begin temperature boundary condition surface_22
    add surface surface_22
    temperature time function = tempRight
  end

  begin contact definition mary
    contact surface surf_12 contains surface_12
    contact surface surf_24 contains surface_24

```

```

begin interaction harry
  surfaces are surf_24 surf_12
  normal tolerance = 10e-5
end
begin enforcement larry
  conductance coefficient =2000
  enforcement for energy = gap_conductance
end
update search every 1 steps
end

use finite element model mesh_calagio model coordinates are model_coordinates
nonlinear solution strategy = newton
use dof averaged nonlinear residual
accept solution after maximum nonlinear iterations = true
use linear solver solve_temperature_aztec

end Aria region Aria

begin adagio region adagio
  use finite element model mesh_calagio model coordinates are model_coordinates

  ### output description ###
  begin Results Output output_adagio
    Database Name = gapClosure.e
    At Step 0, Increment = 1
    nodal Variables = temperature as temp
    nodal Variables = velocity as vel
    nodal Variables = displacement as displ
    element Variables = stress as stress
    global Variables = timestep as timestep
    global variables = external_energy as ExternalEnergy
    global variables = internal_energy as InternalEnergy
    global variables = kinetic_energy as KineticEnergy
    global variables = momentum as Momentum
  end results output output_adagio

  ### definition of BCs ###
  begin fixed displacement
    surface = surface_14 surface_22
    components = x
  end fixed displacement

  begin fixed displacement
    surface = surface_11 surface_21
    components = y
  end fixed displacement

```

```

# Contact

begin contact definition
  search = dash
  contact surface surf_12 contains surface_12
  contact surface surf_24 contains surface_24
  begin interaction
    master = surf_24
    slave = surf_12
    normal tolerance = 2.5e-4
    tangential tolerance = 1e-2
    capture tolerance = 0.25e-4
  end
  enforcement = al
end

### -----###
### Solver definition ###
### -----###

begin solver

begin Loadstep predictor
  type = secant
  slip scale factor = 0.0
end

begin control contact
  target relative residual = 1.0e-7
  target residual = 1.0e-12
  maximum iterations = 10
  minimum iterations = 0
end

Begin cg
  target relative residual = 1.0e-10
  target residual = 1.0e-15
  Maximum Iterations = 500
  Minimum Iterations = 0
end

end

end adagio region adagio

end procedure Acca_Procedure

```

```
end sierra gapClosure
```

## 7.5. ESTACK REGRESSION TEST

### 7.5.1. Problem Statement

This is a test of solving a simple one-dimensional thermal diffusion problem with Dirichlet BCs. The test problem is shown schematically in Figure. Although the problem is one-dimensional we solve the problem in a three-dimensional setting. Once the diffusion problem has been solved numerically the temperature result is postprocessed to obtain a comparison with the analytical result and the distribution of diffusive heat flux. This test input also demonstrates the use tabular function and Encore function material property specification in Aria.

### 7.5.2. Input File

```
#
Begin Sierra Aria_Presto_example

Define Direction Y_Axis With Vector 0.0 1.0 0.0

Begin Definition For Function Delta
  Type is Piecewise Linear
  Ordinate is Displacement
  Abscissa is Time
  Begin Values
    0.000E+00 0.00000E+00
    1.400E-04 1.40098E-03
    2.800E-04 2.80392E-03
    4.200E-04 4.20883E-03
    5.600E-04 5.61571E-03
    7.000E-04 7.02456E-03
    8.400E-04 8.43538E-03
    9.800E-04 9.84818E-03
    1.120E-03 1.12630E-02
    1.260E-03 1.26797E-02
    1.400E-03 1.40985E-02
    1.540E-03 1.55192E-02
    1.680E-03 1.69419E-02
    1.820E-03 1.83666E-02
    1.960E-03 1.97933E-02
    2.100E-03 2.12221E-02
    2.240E-03 2.26528E-02
    2.380E-03 2.40855E-02
```

2.520E-03 2.55202E-02  
2.660E-03 2.69569E-02  
2.800E-03 2.83957E-02  
2.940E-03 2.98364E-02  
3.080E-03 3.12792E-02  
3.220E-03 3.27240E-02  
3.360E-03 3.41709E-02  
3.500E-03 3.56197E-02  
3.640E-03 3.70706E-02  
3.780E-03 3.85235E-02  
3.920E-03 3.99785E-02  
4.060E-03 4.14354E-02  
4.200E-03 4.28945E-02  
4.340E-03 4.43556E-02  
4.480E-03 4.58187E-02  
4.620E-03 4.72838E-02  
4.760E-03 4.87511E-02  
4.900E-03 5.02204E-02  
5.040E-03 5.16917E-02  
5.180E-03 5.31651E-02  
5.320E-03 5.46406E-02  
5.460E-03 5.61181E-02  
5.600E-03 5.75977E-02  
5.740E-03 5.90794E-02  
5.880E-03 6.05631E-02  
6.020E-03 6.20489E-02  
6.160E-03 6.35368E-02  
6.300E-03 6.50268E-02  
6.440E-03 6.65189E-02  
6.580E-03 6.80131E-02  
6.720E-03 6.95094E-02  
6.860E-03 7.10077E-02  
7.000E-03 7.25082E-02  
7.140E-03 7.40107E-02  
7.280E-03 7.55154E-02  
7.420E-03 7.70222E-02  
7.560E-03 7.85311E-02  
7.700E-03 8.00421E-02  
7.840E-03 8.15552E-02  
7.980E-03 8.30704E-02  
8.120E-03 8.45878E-02  
8.260E-03 8.61073E-02  
8.400E-03 8.76289E-02  
8.540E-03 8.91526E-02  
8.680E-03 9.06785E-02  
8.820E-03 9.22065E-02  
8.960E-03 9.37367E-02

9.100E-03 9.52690E-02  
9.240E-03 9.68035E-02  
9.380E-03 9.83401E-02  
9.520E-03 9.98788E-02  
9.660E-03 1.01420E-01  
9.800E-03 1.02963E-01  
9.940E-03 1.04508E-01  
1.008E-02 1.06055E-01  
1.022E-02 1.07605E-01  
1.036E-02 1.09157E-01  
1.050E-02 1.10711E-01  
1.064E-02 1.12267E-01  
1.078E-02 1.13825E-01  
1.092E-02 1.15385E-01  
1.106E-02 1.16948E-01  
1.120E-02 1.18513E-01  
1.134E-02 1.20080E-01  
1.148E-02 1.21649E-01  
1.162E-02 1.23220E-01  
1.176E-02 1.24794E-01  
1.190E-02 1.26370E-01  
1.204E-02 1.27948E-01  
1.218E-02 1.29528E-01  
1.232E-02 1.31111E-01  
1.246E-02 1.32695E-01  
1.260E-02 1.34282E-01  
1.274E-02 1.35871E-01  
1.288E-02 1.37463E-01  
1.302E-02 1.39056E-01  
1.316E-02 1.40652E-01  
1.330E-02 1.42250E-01  
1.344E-02 1.43850E-01  
1.358E-02 1.45453E-01  
1.372E-02 1.47058E-01  
1.386E-02 1.48665E-01  
1.400E-02 1.50274E-01  
1.414E-02 1.51885E-01  
1.428E-02 1.53499E-01  
1.442E-02 1.55115E-01  
1.456E-02 1.56733E-01  
1.470E-02 1.58354E-01  
1.484E-02 1.59977E-01  
1.498E-02 1.61602E-01  
1.512E-02 1.63229E-01  
1.526E-02 1.64859E-01  
1.540E-02 1.66491E-01  
1.554E-02 1.68125E-01



1.568E-02 1.69762E-01  
1.582E-02 1.71400E-01  
1.596E-02 1.73042E-01  
1.610E-02 1.74685E-01  
1.624E-02 1.76331E-01  
1.638E-02 1.77979E-01  
1.652E-02 1.79629E-01  
1.666E-02 1.81282E-01  
1.680E-02 1.82937E-01  
1.694E-02 1.84594E-01  
1.708E-02 1.86253E-01  
1.722E-02 1.87915E-01  
1.736E-02 1.89580E-01  
1.750E-02 1.91246E-01  
1.764E-02 1.92915E-01  
1.778E-02 1.94586E-01  
1.792E-02 1.96260E-01  
1.806E-02 1.97936E-01  
1.820E-02 1.99614E-01  
1.834E-02 2.01295E-01  
1.848E-02 2.02978E-01  
1.862E-02 2.04663E-01  
1.876E-02 2.06351E-01  
1.890E-02 2.08041E-01  
1.904E-02 2.09733E-01  
1.918E-02 2.11428E-01  
1.932E-02 2.13125E-01  
1.946E-02 2.14825E-01  
1.960E-02 2.16527E-01  
1.974E-02 2.18231E-01  
1.988E-02 2.19938E-01  
2.002E-02 2.21647E-01  
2.016E-02 2.23359E-01  
2.030E-02 2.25072E-01  
2.044E-02 2.26789E-01  
2.058E-02 2.28507E-01  
2.072E-02 2.30229E-01  
2.086E-02 2.31952E-01  
2.100E-02 2.33678E-01  
2.114E-02 2.35406E-01  
2.128E-02 2.37137E-01  
2.142E-02 2.38870E-01  
2.156E-02 2.40606E-01  
2.170E-02 2.42344E-01  
2.184E-02 2.44085E-01  
2.198E-02 2.45828E-01  
2.212E-02 2.47573E-01

2.226E-02 2.49321E-01  
2.240E-02 2.51071E-01  
2.254E-02 2.52824E-01  
2.268E-02 2.54579E-01  
2.282E-02 2.56337E-01  
2.296E-02 2.58097E-01  
2.310E-02 2.59859E-01  
2.324E-02 2.61624E-01  
2.338E-02 2.63392E-01  
2.352E-02 2.65162E-01  
2.366E-02 2.66934E-01  
2.380E-02 2.68709E-01  
2.394E-02 2.70487E-01  
2.408E-02 2.72267E-01  
2.422E-02 2.74049E-01  
2.436E-02 2.75834E-01  
2.450E-02 2.77621E-01  
2.464E-02 2.79411E-01  
2.478E-02 2.81204E-01  
2.492E-02 2.82999E-01  
2.506E-02 2.84796E-01  
2.520E-02 2.86596E-01  
2.534E-02 2.88399E-01  
2.548E-02 2.90204E-01  
2.562E-02 2.92011E-01  
2.576E-02 2.93821E-01  
2.590E-02 2.95634E-01  
2.604E-02 2.97449E-01  
2.618E-02 2.99267E-01  
2.632E-02 3.01087E-01  
2.646E-02 3.02910E-01  
2.660E-02 3.04735E-01  
2.674E-02 3.06563E-01  
2.688E-02 3.08393E-01  
2.702E-02 3.10226E-01  
2.716E-02 3.12062E-01  
2.730E-02 3.13900E-01  
2.744E-02 3.15741E-01  
2.758E-02 3.17584E-01  
2.772E-02 3.19430E-01  
2.786E-02 3.21279E-01  
2.800E-02 3.23130E-01  
2.814E-02 3.24983E-01  
2.828E-02 3.26840E-01  
2.842E-02 3.28699E-01  
2.856E-02 3.30560E-01  
2.870E-02 3.32424E-01

2.884E-02 3.34291E-01  
2.898E-02 3.36160E-01  
2.912E-02 3.38032E-01  
2.926E-02 3.39907E-01  
2.940E-02 3.41784E-01  
2.954E-02 3.43664E-01  
2.968E-02 3.45546E-01  
2.982E-02 3.47431E-01  
2.996E-02 3.49319E-01  
3.010E-02 3.51209E-01  
3.024E-02 3.53102E-01  
3.038E-02 3.54998E-01  
3.052E-02 3.56896E-01  
3.066E-02 3.58797E-01  
3.080E-02 3.60701E-01  
3.094E-02 3.62607E-01  
3.108E-02 3.64516E-01  
3.122E-02 3.66428E-01  
3.136E-02 3.68342E-01  
3.150E-02 3.70259E-01  
3.164E-02 3.72179E-01  
3.178E-02 3.74101E-01  
3.192E-02 3.76027E-01  
3.206E-02 3.77954E-01  
3.220E-02 3.79885E-01  
3.234E-02 3.81818E-01  
3.248E-02 3.83754E-01  
3.262E-02 3.85692E-01  
3.276E-02 3.87634E-01  
3.290E-02 3.89578E-01  
3.304E-02 3.91525E-01  
3.318E-02 3.93474E-01  
3.332E-02 3.95426E-01  
3.346E-02 3.97381E-01  
3.360E-02 3.99339E-01  
3.374E-02 4.01299E-01  
3.388E-02 4.03263E-01  
3.402E-02 4.05229E-01  
3.416E-02 4.07197E-01  
3.430E-02 4.09169E-01  
3.444E-02 4.11143E-01  
3.458E-02 4.13120E-01  
3.472E-02 4.15100E-01  
3.486E-02 4.17082E-01  
3.500E-02 4.19068E-01  
3.514E-02 4.21056E-01  
3.528E-02 4.23047E-01

3.542E-02 4.25040E-01  
3.556E-02 4.27037E-01  
3.570E-02 4.29036E-01  
3.584E-02 4.31038E-01  
3.598E-02 4.33043E-01  
3.612E-02 4.35050E-01  
3.626E-02 4.37061E-01  
3.640E-02 4.39074E-01  
3.654E-02 4.41090E-01  
3.668E-02 4.43109E-01  
3.682E-02 4.45131E-01  
3.696E-02 4.47156E-01  
3.710E-02 4.49183E-01  
3.724E-02 4.51213E-01  
3.738E-02 4.53246E-01  
3.752E-02 4.55282E-01  
3.766E-02 4.57321E-01  
3.780E-02 4.59363E-01  
3.794E-02 4.61407E-01  
3.808E-02 4.63455E-01  
3.822E-02 4.65505E-01  
3.836E-02 4.67558E-01  
3.850E-02 4.69614E-01  
3.864E-02 4.71673E-01  
3.878E-02 4.73735E-01  
3.892E-02 4.75800E-01  
3.906E-02 4.77867E-01  
3.920E-02 4.79938E-01  
3.934E-02 4.82011E-01  
3.948E-02 4.84087E-01  
3.962E-02 4.86167E-01  
3.976E-02 4.88249E-01  
3.990E-02 4.90334E-01  
4.004E-02 4.92422E-01  
4.018E-02 4.94512E-01  
4.032E-02 4.96606E-01  
4.046E-02 4.98703E-01  
4.060E-02 5.00803E-01  
4.074E-02 5.02905E-01  
4.088E-02 5.05011E-01  
4.102E-02 5.07119E-01  
4.116E-02 5.09231E-01  
4.130E-02 5.11345E-01  
4.144E-02 5.13462E-01  
4.158E-02 5.15583E-01  
4.172E-02 5.17706E-01  
4.186E-02 5.19832E-01

4.200E-02 5.21962E-01  
4.214E-02 5.24094E-01  
4.228E-02 5.26229E-01  
4.242E-02 5.28367E-01  
4.256E-02 5.30508E-01  
4.270E-02 5.32653E-01  
4.284E-02 5.34800E-01  
4.298E-02 5.36950E-01  
4.312E-02 5.39103E-01  
4.326E-02 5.41260E-01  
4.340E-02 5.43419E-01  
4.354E-02 5.45581E-01  
4.368E-02 5.47746E-01  
4.382E-02 5.49915E-01  
4.396E-02 5.52086E-01  
4.410E-02 5.54261E-01  
4.424E-02 5.56438E-01  
4.438E-02 5.58619E-01  
4.452E-02 5.60802E-01  
4.466E-02 5.62989E-01  
4.480E-02 5.65179E-01  
4.494E-02 5.67371E-01  
4.508E-02 5.69567E-01  
4.522E-02 5.71766E-01  
4.536E-02 5.73968E-01  
4.550E-02 5.76173E-01  
4.564E-02 5.78382E-01  
4.578E-02 5.80593E-01  
4.592E-02 5.82807E-01  
4.606E-02 5.85025E-01  
4.620E-02 5.87245E-01  
4.634E-02 5.89469E-01  
4.648E-02 5.91696E-01  
4.662E-02 5.93926E-01  
4.676E-02 5.96159E-01  
4.690E-02 5.98395E-01  
4.704E-02 6.00634E-01  
4.718E-02 6.02877E-01  
4.732E-02 6.05122E-01  
4.746E-02 6.07371E-01  
4.760E-02 6.09623E-01  
4.774E-02 6.11878E-01  
4.788E-02 6.14136E-01  
4.802E-02 6.16398E-01  
4.816E-02 6.18662E-01  
4.830E-02 6.20930E-01  
4.844E-02 6.23201E-01

4.858E-02 6.25475E-01  
4.872E-02 6.27752E-01  
4.886E-02 6.30033E-01  
4.900E-02 6.32316E-01  
4.914E-02 6.34603E-01  
4.928E-02 6.36893E-01  
4.942E-02 6.39186E-01  
4.956E-02 6.41483E-01  
4.970E-02 6.43783E-01  
4.984E-02 6.46085E-01  
4.998E-02 6.48392E-01  
5.012E-02 6.50701E-01  
5.026E-02 6.53014E-01  
5.040E-02 6.55329E-01  
5.054E-02 6.57648E-01  
5.068E-02 6.59971E-01  
5.082E-02 6.62296E-01  
5.096E-02 6.64625E-01  
5.110E-02 6.66957E-01  
5.124E-02 6.69293E-01  
5.138E-02 6.71631E-01  
5.152E-02 6.73973E-01  
5.166E-02 6.76318E-01  
5.180E-02 6.78667E-01  
5.194E-02 6.81019E-01  
5.208E-02 6.83374E-01  
5.222E-02 6.85732E-01  
5.236E-02 6.88094E-01  
5.250E-02 6.90459E-01  
5.264E-02 6.92827E-01  
5.278E-02 6.95199E-01  
5.292E-02 6.97574E-01  
5.306E-02 6.99952E-01  
5.320E-02 7.02334E-01  
5.334E-02 7.04719E-01  
5.348E-02 7.07107E-01  
5.362E-02 7.09498E-01  
5.376E-02 7.11893E-01  
5.390E-02 7.14292E-01  
5.404E-02 7.16693E-01  
5.418E-02 7.19098E-01  
5.432E-02 7.21507E-01  
5.446E-02 7.23919E-01  
5.460E-02 7.26334E-01  
5.474E-02 7.28752E-01  
5.488E-02 7.31174E-01  
5.502E-02 7.33600E-01

5.516E-02 7.36028E-01  
5.530E-02 7.38461E-01  
5.544E-02 7.40896E-01  
5.558E-02 7.43335E-01  
5.572E-02 7.45777E-01  
5.586E-02 7.48223E-01  
5.600E-02 7.50673E-01  
5.614E-02 7.53125E-01  
5.628E-02 7.55581E-01  
5.642E-02 7.58041E-01  
5.656E-02 7.60504E-01  
5.670E-02 7.62970E-01  
5.684E-02 7.65440E-01  
5.698E-02 7.67913E-01  
5.712E-02 7.70390E-01  
5.726E-02 7.72871E-01  
5.740E-02 7.75354E-01  
5.754E-02 7.77842E-01  
5.768E-02 7.80332E-01  
5.782E-02 7.82826E-01  
5.796E-02 7.85324E-01  
5.810E-02 7.87825E-01  
5.824E-02 7.90330E-01  
5.838E-02 7.92838E-01  
5.852E-02 7.95350E-01  
5.866E-02 7.97865E-01  
5.880E-02 8.00384E-01  
5.894E-02 8.02906E-01  
5.908E-02 8.05432E-01  
5.922E-02 8.07962E-01  
5.936E-02 8.10494E-01  
5.950E-02 8.13031E-01  
5.964E-02 8.15571E-01  
5.978E-02 8.18115E-01  
5.992E-02 8.20662E-01  
6.006E-02 8.23212E-01  
6.020E-02 8.25767E-01  
6.034E-02 8.28325E-01  
6.048E-02 8.30886E-01  
6.062E-02 8.33451E-01  
6.076E-02 8.36020E-01  
6.090E-02 8.38592E-01  
6.104E-02 8.41168E-01  
6.118E-02 8.43747E-01  
6.132E-02 8.46330E-01  
6.146E-02 8.48917E-01  
6.160E-02 8.51507E-01

6.174E-02 8.54101E-01  
6.188E-02 8.56699E-01  
6.202E-02 8.59300E-01  
6.216E-02 8.61905E-01  
6.230E-02 8.64513E-01  
6.244E-02 8.67125E-01  
6.258E-02 8.69741E-01  
6.272E-02 8.72361E-01  
6.286E-02 8.74984E-01  
6.300E-02 8.77611E-01  
6.314E-02 8.80241E-01  
6.328E-02 8.82875E-01  
6.342E-02 8.85513E-01  
6.356E-02 8.88155E-01  
6.370E-02 8.90800E-01  
6.384E-02 8.93449E-01  
6.398E-02 8.96102E-01  
6.412E-02 8.98758E-01  
6.426E-02 9.01418E-01  
6.440E-02 9.04082E-01  
6.454E-02 9.06750E-01  
6.468E-02 9.09421E-01  
6.482E-02 9.12096E-01  
6.496E-02 9.14775E-01  
6.510E-02 9.17457E-01  
6.524E-02 9.20144E-01  
6.538E-02 9.22834E-01  
6.552E-02 9.25528E-01  
6.566E-02 9.28225E-01  
6.580E-02 9.30927E-01  
6.594E-02 9.33632E-01  
6.608E-02 9.36341E-01  
6.622E-02 9.39054E-01  
6.636E-02 9.41770E-01  
6.650E-02 9.44491E-01  
6.664E-02 9.47215E-01  
6.678E-02 9.49943E-01  
6.692E-02 9.52675E-01  
6.706E-02 9.55410E-01  
6.720E-02 9.58150E-01  
6.734E-02 9.60893E-01  
6.748E-02 9.63640E-01  
6.762E-02 9.66391E-01  
6.776E-02 9.69146E-01  
6.790E-02 9.71905E-01  
6.804E-02 9.74667E-01  
6.818E-02 9.77434E-01



```

        6.832E-02 9.80204E-01
        6.846E-02 9.82978E-01
        6.860E-02 9.85757E-01
        6.874E-02 9.88539E-01
        6.888E-02 9.91325E-01
        6.902E-02 9.94114E-01
        6.916E-02 9.96908E-01
        6.930E-02 9.99706E-01
        6.944E-02 1.00251E+00
        6.958E-02 1.00531E+00
        6.972E-02 1.00812E+00
        6.986E-02 1.01094E+00
        7.000E-02      1.01375E+00
End Values
End Definition For Function Delta

begin definition for function TEMPERATURE
    type is piecewise linear
    ordinate is temperature
    abscissa is time
    begin values
        0.00  1255.4
        1.00  1255.4
    end values
end definition for function TEMPERATURE

begin definition for function THERMAL_STRAIN
    type is piecewise linear
    ordinate is strain
    abscissa is temperature
    begin values
        200.0  0.0
        3000.0  0.0
    end values
end definition for function THERMAL_STRAIN

Begin Property Specification For Material Resistor
    density      = 8.0E-4
    thermal strain function = THERMAL_STRAIN
    begin parameters for model elastic
        youngs modulus = 200.0E3 $ MPa
        poissons ratio  = 0.305
    End
End

Begin Aria Material Resistor
    Electric Displacement  = Basic

```

```

    Electrical Permittivity = Constant Kappa=3e-10 # N/V^2
    Electrical Resistivity  = Constant Rho=2e2 # Ohm-mm
End

```

```

Begin Property Specification For Material Metal
    density          = 8.0E-4
    thermal strain function = THERMAL_STRAIN
    begin parameters for model elastic
        youngs modulus = 200.0E3 $ MPa
        poissons ratio  = 0.305
    End
End

```

```

Begin Aria Material Metal
    Electric Displacement    = Basic
    Electrical Permittivity = Constant Kappa=1e-9 # N/V^2
    Electrical Resistivity  = Constant Rho=2e-5 # Ohm-mm
End

```

```

begin property specification for material dielectric
    density          = 8.0E-4
    thermal strain function = THERMAL_STRAIN
    begin parameters for model elastic
        youngs modulus = 200.0E3 $ MPa
        poissons ratio  = 0.305
    end
end

```

```

Begin Aria Material Dielectric
    Electric Displacement    = Basic
    Electrical Permittivity = Constant Kappa=3e-11 # N/V^2
    Electrical Resistivity  = Constant Rho=1e13 # Ohm-mm
End

```

```

begin finite element model mesh1
    Database Name = estack.g
    Database Type = exodusII

    begin parameters for block block_1
        material resistor
        model = elastic
        hourglass stiffness = 0.05
        hourglass viscosity = 0.03
    end parameters for block block_1

```

```

begin parameters for block block_2
  material metal
  model = elastic
  hourglass stiffness = 0.05
  hourglass viscosity = 0.03
end parameters for block block_2

begin parameters for block block_3
  material dielectric
  model = elastic
  hourglass stiffness = 0.05
  hourglass viscosity = 0.03
end parameters for block block_3

begin parameters for block block_4
  material metal
  model = elastic
  hourglass stiffness = 0.05
  hourglass viscosity = 0.03
end parameters for block block_4

end finite element model mesh1

# Trilinos direct equation solver for Aria
Begin Trilinos Equation Solver Direct_Solver
  Solution Method = amesos-superlu
End

Begin Procedure The_Procedure

  Begin Solution Control Description
    # Define solution control for advancing and transferring forces and
    # displacements between Aria and Adagio
    Begin System Main
      Begin Transient MyTransient
        Transfer Presto_to_Aria
        Advance AriaRegion
        Begin Subcycle PrestoSubcycle
          Advance PrestoRegion
        End
      End
    End
  End

  # Time-stepping parameters
  Begin Parameters for Transient MyTransient
    Start Time      = 0.0

```

```

time step style noclip
Termination Time = 0.07
# Parameters for Aria region: fluid mechanics region
Begin Parameters for Aria Region AriaRegion
    Initial Time Step Size = 1e-3
    Time Step Variation    = Fixed
End
# Parameters for Adagio region: solid mechanics region
Begin Parameters for Presto Region PrestoRegion
    initial time step = 1.0e-6
    time step scale factor = 1.0
    time step increase factor = 10.
    # step interval = 500
End
End

End Solution Control Description

Begin Transfer Presto_to_Aria
Copy Volume Nodes from PrestoRegion to AriaRegion
Send Field displacement State New to Solution->Mesh_Displacements State New
End

Begin Presto Region PrestoRegion

    use finite element model mesh1
    Begin Prescribed Temperature
        function = TEMPERATURE
        scale factor = 1.0
        include all blocks
    End

    ### output description ###
    begin Results Output output_presto
        Database Name = estack_presto.e
        Database Type = exodusII
        At Time 0.0, Increment = 1.0E-3
        nodal Variables = force_external as f_ext
        nodal Variables = force_internal as f_int
        nodal Variables = velocity as vel
        nodal Variables = acceleration as accl
        nodal Variables = displacement as displ
        nodal Variables = temperature as temp
        element Variables = stress as stress
        global Variables = timestep as timestep
        global variables = external_energy as ExternalEnergy
        global variables = internal_energy as InternalEnergy

```

```

        global variables = kinetic_energy as KineticEnergy
        global variables = momentum as Momentum
    End

    ### definition of boundary conditions ###

    begin fixed displacement
        node set      = nodelist_1
        components = x
    End

    begin fixed displacement
        node set      = nodelist_2
        components = z
    End

    begin fixed displacement
        node set      = nodelist_3
        components = y
    End

    Begin Prescribed Displacement
        Node set      = nodelist_4
        direction      = y_axis
        function        = delta
        scale factor = 1.0
    End

    End Presto Region PrestoRegion

    Begin Aria Region AriaRegion
    Use Finite Element Model mesh1
    Use Linear Solver Direct_Solver
    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1e-6

    EQ Voltage for Voltage on all_blocks using Q1 with DIFF
    IC const          at block_1      Voltage = 5
    BC const dirichlet at nodelist_3 Voltage = 0 # ground
    BC const dirichlet at nodelist_4 Voltage = 10 # prescribed

    EQ Mesh for Mesh_Displacements on all_blocks using Q1 with XFER

    Begin Results Output The_Output
        Database Name = estack_aria.e
        At step 0, increment = 1

```

```

    Title Aria-Presto electro-mechanical coupling
    Nodal Variables = solution->Voltage          as V
    Nodal Variables = solution->Mesh_Displacements as Disp
End

    End Aria Region AriaRegion

End Procedure The_Procedure

End Sierra Aria_Presto_example

```

## 7.6. TV REGRESSION TEST

### 7.6.1. Problem Statement

This is a test of solving a simple one-dimensional thermal diffusion problem with Dirichlet BCs. The test problem is shown schematically in Figure. Although the problem is one-dimensional we solve the problem in a three-dimensional setting. Once the diffusion problem has been solved numerically the temperature result is postprocessed to obtain a comparison with the analytical result and the distribution of diffusive heat flux. This test input also demonstrates the use tabular function and Encore function material property specification in Aria.

### 7.6.2. Input File

```

#
#
Begin Sierra Slump_Test

    Begin Aria Material Bar
        Thermal Conductivity    = Constant k = 1.0
        Electrical Conductivity = Constant sigma = 1.0
        heat conduction = fouriers_law
        current density = ohms_law
    End

    Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-umfpack
    End

    Begin Finite Element Model The_Model
        Database Name = mesh3d.g
        Begin Parameters For Block block_1
Material Bar

```

```

        End
    End

    Begin Procedure The_Procedure

        begin solution control description
    use system main

    begin system main

        begin sequential mysolveblk
            advance Voltage_Region
            transfer VtoT
            advance Temperature_Region
        end

    end system main

        end solution control description

        begin transfer VtoT
    copy volume nodes from Voltage_Region to Temperature_Region
    send field solution->VOLTAGE state old to solution->VOLTAGE state new
        end transfer VtoT

        Begin Aria Region Voltage_Region

    Use Finite Element Model The_Model

    Use Linear Solver Direct_Solver

    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-6
    Nonlinear Relaxation Factor = 1.0

    # Sideset 1 : x = x_min
    # Sideset 2 : x = x_max
    # Sideset 3 : y = y_min
    # Sideset 4 : y = y_max
    # Sideset 5 : z = z_min
    # Sideset 6 : z = z_max
    # Sideset 10: y and z surfaces

    EQ Current For Voltage On Block_1 Using Q1 With Diff
    IC Const on block_1 Voltage = 0.0
    BC Const Dirichlet at surface_1 Voltage = 10 # fat/low

```

```

BC Const Dirichlet at surface_2 Voltage = 0 # Thin/High End

Begin Results Output Label V_Output
  Database Name = v.e
  At Step 0, Increment is 1
  Title TV Test - V Region
  Nodal Variables = solution->VOLTAGE as V
End

  End

  Begin Aria Region Temperature_Region

Use Finite Element Model The_Model

Use Linear Solver Direct_Solver

Nonlinear Solution Strategy = Newton
Maximum Nonlinear Iterations = 10
Nonlinear Residual Tolerance = 1.0e-6
Nonlinear Relaxation Factor = 1.0

# Sideset 1 : x = x_min
# Sideset 2 : x = x_max
# Sideset 3 : y = y_min
# Sideset 4 : y = y_max
# Sideset 5 : z = z_min
# Sideset 6 : z = z_max
# Sideset 10: y and z surfaces

EQ Current For Voltage On Block_1 Using Q1 With xfer

EQ Energy For Temperature On Block_1 Using Q1 With Diff Src
IC Const on block_1 Temperature = 298
BC Flux for Energy on surface_3 = Nat_Conv H = 20 T_ref = 298
BC Flux for Energy on surface_4 = Nat_Conv H = 0.2 T_ref = 298
Source for Energy on Block_1 = Joule_Heating

Begin Results Output Label T_Output
  Database Name = tv.e
  At Step 0, Increment is 1
  Title TV Test - T Region
  Nodal Variables = solution->VOLTAGE as V
  Nodal Variables = solution->TEMPERATURE as T
End

  End

```



End

End



# BIBLIOGRAPHY

- [1] Gerald W. Wellman. Mapvar: a computer program to transfer solution data between finite element meshes. SAND 1999-0466, Sandia National Laboratories, Albuquerque, NM, USA, March 1999. [1.1](#)
- [2] The SNTTools Project. [SNTTools SourceForge Project](#). Online. [2.3](#)



# INDEX

## A

Abort If Field Not Defined On Copy Transfer Send Or  
Receive Object, [99](#), [100](#)  
Abscissa, [39](#), [40](#)  
Abscissa Offset, [39](#), [41](#)  
Abscissa Scale, [39](#), [41](#)  
Adapt, [61](#), [62](#), [65](#), [66](#), [70](#), [74](#), [78](#), [79](#)  
Adaptivity, [61](#), [66](#), [78](#)  
Advance, [65](#), [67](#), [70](#), [71](#), [74](#), [75](#), [78](#), [79](#), [82](#), [87](#)  
Alias, [28](#), [29](#)  
All Fields, [99](#), [100](#)  
At Discontinuity Evaluate To, [39](#), [41](#)

## B

barOneWayCoupleDifferentMesh  
Statement, [119](#)  
barOneWayCoupleFromDifferentMesh  
Input, [119](#)  
barOneWayCoupleFromFile  
Input, [115](#)  
Statement, [115](#)  
barOneWayCoupleTransfer  
Input, [120](#)  
Statement, [120](#)

## C

Column Titles, [39](#), [41](#)  
Component Separator Character, [28](#), [30](#)  
Compute Indicator On, [61](#), [62](#), [65](#), [67](#), [70](#), [71](#), [74](#), [75](#), [78](#), [79](#)  
Converged When, [83](#), [84](#), [87](#)  
Coordinate System, [28](#), [30](#)  
Copy, [99](#), [100](#)  
Create, [28](#), [30](#)  
Create Element Field, [109](#), [110](#)  
Create Nodal Field, [109](#), [111](#)

## D

Data File, [39](#), [42](#)  
Database Name, [28](#), [31](#)  
Database Type, [29](#), [31](#)  
Debug, [39](#), [42](#)  
Decomposition Method, [29](#), [31](#)  
Definition For Function, [39](#)  
Differentiate Expression, [39](#), [42](#)  
Distance Function Is Closest Receive Node To Send  
Centroid, [99](#), [101](#)

## E

estack  
Input, [136](#)  
Statement, [136](#)  
Evaluate Expression, [39](#), [43](#)  
Evaluate From, [39](#), [45](#)  
Event, [61](#), [62](#), [65](#), [67](#), [70](#), [71](#), [74](#), [75](#), [78](#), [80](#), [82](#), [87](#)  
Exclude Ghosted, [99](#), [101](#)  
Execute Postprocessor Group, [61](#), [63](#), [66](#), [68](#), [70](#), [72](#), [74](#), [76](#),  
[78](#), [80](#)  
Expression Variable:, [39](#), [45](#), [46](#)  
Expressions, [40](#)

## F

Finite Element Model, [28](#)  
Fixed Time, [109](#), [111](#)  
From, [99](#), [101](#)

## G

gapClosure  
Input, [125](#)  
Statement, [125](#)  
Gauss Point Integration Order, [99](#), [101](#)  
Global Constants, [36](#)  
Global Id Mapping Backward Compatibility, [29](#), [32](#)  
Gravity Vector, [36](#), [37](#)

## H

Heartbeat, [110](#)  
History Output, [110](#)

## I

Ideal Gas Constant, [36](#), [37](#)  
Include All Blocks, [33](#), [34](#)  
Incremental Number Of Steps, [83](#), [84](#)  
Indicatemarkadapt, [61](#), [63](#), [66](#), [68](#), [70](#), [72](#), [74](#), [76](#), [78](#), [80](#)  
Initial Deltat, [83](#), [84](#), [87](#)  
Initialize, [60](#), [82](#)  
Input\_Output Region, [109](#)  
Interpolate, [99](#), [102](#)  
Interpolation Function, [99](#), [102](#)  
Inversion Aversion Exponent, [33](#), [34](#)  
Inversion Aversion Stiffness, [34](#), [35](#)  
Inversion Aversion Transition Jacobian, [34](#), [35](#)  
Involve, [66](#), [68](#), [70](#), [72](#), [74](#), [76](#), [78](#), [80](#), [82](#), [83](#), [88](#)

K  
K-E Turbulence Model Parameter, 36, 37  
K-W Turbulence Model Parameter, 36, 38

L  
Les Turbulence Model Parameter, 36, 38  
Light Speed, 36, 38  
Local Coordinate System, 34, 35

M  
Mark, 61, 63, 66, 69, 70, 73, 74, 77, 78, 81  
Markadapt, 61, 64, 66, 69, 70, 73, 74, 77, 78, 81  
Material, 34, 35  
Material =, 34, 36

N  
Nodes Outside Region, 99, 103  
Nonlinear, 66, 70, 78  
Number Of Adaptivity Steps, 83, 85, 88  
Number Of Steps, 83, 85, 88

O  
Offset Time, 109, 111  
Omit Block, 29, 32  
Omit Volume, 29, 32  
Ordinate, 40, 46  
Ordinate Offset, 40, 46  
Ordinate Scale, 40, 47  
Output, 61, 64, 66, 69, 70, 73, 74, 77, 78, 81, 88

P  
Parameters For, 60, 83  
Parameters For Aria Region, 83  
Parameters For Block, 29, 33  
Parameters For Phase, 29  
Parameters For Surface, 29  
Periodicity Time, 109, 112  
Phase, 34, 36  
Planck Constant, 37, 38

R  
Receive Blocks, 99  
Reinitialize Transient, 83, 85, 88  
Remove Block, 34, 36  
Restart Data, 110  
Restart Time, 49  
Results Output, 110

S  
Scale By, 40, 47  
Search Coordinate Field, 99, 103  
Search Geometric Tolerance, 99, 104  
Search Surface Gap Tolerance, 99, 104  
Search Type, 99, 105  
Select One Receiver For Each Send Object, 99, 105

Select One Unique Receiver For Each Send Object, 99, 105  
Send, 99, 105  
Send Block, 99, 106  
Send Blocks, 99  
Send Field, 99, 106  
Sequential, 61, 78  
Simulation Max Global Iterations, 61, 64, 89  
Simulation Start Time, 61, 64, 89  
Simulation Termination Time, 61, 65, 89  
Solution Control Description, 60  
Start Time, 83, 85, 89, 109, 113  
Stefan Boltzmann Constant, 37, 39  
Subcycle, 66, 70, 74  
System, 60, 61

T  
Termination Time, 83, 85, 90  
Time Scale Factor, 29, 33  
Time Step Quantum, 83, 86, 90  
Time Step Style, 83, 86, 90  
Total Change In Time, 83, 86, 91  
Transfer, 61, 65, 66, 69, 70, 73, 74, 77, 78, 81–83, 91, 98, 99  
Transient, 61, 65  
Turbulence Model, 37, 39  
tv  
    Input, 152  
    Statement, 152  
Type, 40, 47

U  
Use Finite Element Model, 110, 113  
Use Generic Names, 29, 33  
Use Initialize, 61, 65, 91  
Use Material, 29, 33  
Use System, 60, 91

V  
Values, 40, 48

X  
X Offset, 40, 47  
X Scale, 40, 48

Y  
Y Offset, 40, 48  
Y Scale, 40, 48

# DISTRIBUTION

Email—Internal [REDACTED]

Name	Org.	Sandia Email Address
Technical Library	01177	libref@sandia.gov









Sandia  
National  
Laboratories

Sandia National Laboratories  
is a multimission laboratory  
managed and operated by  
National Technology &  
Engineering Solutions of  
Sandia LLC, a wholly owned  
subsidiary of Honeywell  
International Inc., for the U.S.  
Department of Energy's  
National Nuclear Security  
Administration under contract  
DE-NA0003525.