SANDIA REPORT

SAND2018-12008
Unlimited Release
Printed October 18, 2018

SIERRA Multimechanics Module: Aria User Manual – Version 4.50

SIERRA Thermal/Fluid Development Team

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

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.

Sandia National Laboratories

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

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@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

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

SAND2018-12008
Unlimited Release
Printed October 18, 2018

SIERRA Multimechanics Module: Aria User Manual – Version 4.50

SIERRA Thermal/Fluid Development Team

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Abstract

Aria is a Galerkin finite element based program for solving coupled-physics problems described by systems of PDEs and is capable of solving nonlinear, implicit, transient and direct-to-steady state problems in two and three dimensions on parallel architectures. The suite of physics currently supported by Aria includes thermal energy transport, species transport, and electrostatics as well as generalized scalar, vector and tensor transport equations. Additionally, Aria includes support for manufacturing process flows via the incompressible Navier-Stokes equations specialized to a low Reynolds number ($Re < 1$) regime. Enhanced modeling support of manufacturing processing is made possible through use of either arbitrary Lagrangian-Eulerian (ALE) and level set based free and moving boundary tracking in conjunction with quasi-static nonlinear elastic solid mechanics for mesh control. Coupled physics problems are solved in several ways including fully-coupled Newton's method with analytic or numerical sensitivities, fully-coupled Newton-Krylov methods and a loosely-coupled nonlinear iteration about subsets of the system that are solved using combinations of the aforementioned methods. Error estimation, uniform and dynamic h-adaptivity and dynamic load balancing are some of Aria's more advanced capabilities.

Acknowledgments

Aria represents a new generation code implementation of heat transfer algorithms. The mathematical models described herein would never have been implemented without the efforts of the Calore development team: Bob Cochran, Mike Glass, Randy Lober, Chris Newman, Steve Bova and Tolu Okusanya. Bruce Bainbridge, Ben Blackwell, Merlin Decker, Kevin Dowding and Vicente Romero made significant contributions to the development and testing of Calore. Many of the algorithms implemented in Calore were first implemented in COYOTE, and therefore we are indebted to David Gartling, Roy Hogan, and others who contributed to its development. In particular, the algorithms associated with chemistry were developed by Mel Baer.

The basic capabilities of Calore were implemented in Aria by Patrick Notz and Samuel Subia. Further implementation of new capabilities by the Sierra T/F code team is a continuing effort. Additionally, the following Sandia staff have been involved with the Aria project, either in code development, math model definitions, or requirements definition: Steve Bova, Victor Brunini, Jonathan Clausen, Stefan Domino, Lindsay Erickson, Kenneth Franko, David Glaze, Michael Hansen, Matthew Hopkins, Adrian Kopacz, Harry Moffatt, David Noble, Patrick Notz, Tolulope Okusanya, Flint Pierce, Scott Roberts, Ryan Shaw, Samuel Subia, Dan Turner, and Tyler Voskuilen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The SIERRA Multimechanics Module: Aria, henceforth referred to as Aria for brevity, is an application implementing the finite element method (FEM) for solving systems of partial differential equations (PDEs). Foremost, Aria's development targets applications which involve incompressible flow (Navier-Stokes for $Re < 1$). However, the general design of Aria lends itself to the solution of systems of PDEs describing physical processes including energy transport, species transport with reactions, electrostatics and general transport of scalar, vector and tensor quantities in two and three dimensions both transient and direct to steady state. Moreover, different regions of the physical domain (i.e., the input mesh) may have either different materials and/or different collections of physics (viz., PDEs) defined on them. These systems of equations may be solved alone, in a segregated but coupled algorithm ("loosely coupled") or as a single, fully-coupled system. Additionally, Aria can be loosely coupled to the quasi-statics solid mechanics code Adagio using the coupling application Arpeggio.

Aria is able to accommodate meshes that utilize linear and quadratic elements in two and three dimensions. In two dimensions, Aria supports quadrilateral (4 and 9 node) and triangular (3 and 6 node) elements. In three dimensions, Aria supports hexahedral (8 and 27 node) and tetrahedral (4 and 10 node) elements. Moreover, meshes may be comprised of combinations of these elements (i.e., both quadrilateral and triangular elements in two dimensions).

The physical coordinates and mesh displacements are always interpolated in accordance with the input mesh, but other solution degrees of freedom may be interpolated using a lower order basis function. For example, if the input mesh is composed of 9 node (quadratic) quadrilateral elements, then the physical coordinates and mesh displacements (if active) will be interpolated using quadratic basis functions, whereas other degrees of freedom, e.g., temperature or voltage, could use linear shape functions.

Additional information concerning the project may be found at the Aria's home page, Aria Users Homepage [1], and at Aria's Trac wiki web site, Aria Trac Wiki [2]. Both of those web sites currently require access to Sandia's internal restricted network.

1.2 A Brief History of Aria

In many respects, the predecessor code to Aria is the highly successful Goma code. Although none of the Goma code is included in Aria, Goma has inspired many of the algorithm, design and implementation decisions in Aria. Several of the staff members who contributed to Goma have also contributed to Aria, either as developers or users and testers.

Like Goma, Aria's primary nonlinear algorithm is a full Newton-Raphson method with analytical sensitivities. However, Aria takes aspects of the implementation further by adding versatility in how the sensitivities are constructed and providing options for both finite-difference, and automatic differentiation sensitivities.

Furthermore, Aria has a richer suite of nonlinear solver capabilities including matrix-free Newton-Krylov, methods and even loose coupling. Like its muse Goma, Aria includes a large suite of physics and capabilities which can be included in an analysis.

On December 5, 2000 Phil Sackinger initiated the Opera code which was to serve as a test be for what would become Aria. The Opera code was a patterned after the Calore code which is a thermal analysis GFEM code, but differed in many respects. At that time, no applications had been written on top of the Sierra Framework that used multiple degrees of freedom (e.g., velocity AND pressure) or that used multiple element type (e.g., linear and quadratic interpolations). Opera was created as an exploratory code, to determine what changes would need to be made to the Framework and to experiment with the Framework in constructing such an application. During this period, the principle developers of Opera where Phil Sackinger, Sam Subia and Matt Hopkins.

On November 20, 2001 Matt Hopkins made the first commit to the Aria code base, using Opera as the starting point. And thus Aria was born.

1.3 Nonlinear Coupling Strategies in Aria

One of the difficulties with writing broadly applicable computational mechanics software is that developers can't take advantage of specific knowledge of the application domain in order to optimize the algorithm. Thus, in providing generality one sometimes sacrifices efficiency. One place this is evident in multiphysics modeling is in the choice of coupling strategies. While it is well understood that a fully coupled system solved with Newton's method utilizing analytic sensitivities is formally the most robust and correct approach to solving multiphysics applications it is also computationally expensive and complex to implement. Furthermore, while Newton's method has the fastest rate of asymptotic convergence it's domain of convergence is often empirically observed to be smaller than other methods. Lastly, in some applications, certain subsets of the physics may be only weakly coupled so that a loosely coupled approach may be more computationally efficient. To address these concerns while remaining general and flexible Aria offers a number of options for nonlinear solution strategies and physics coupling.

In defining a problem in Aria, users configure one or more Regions. Each Region consists of one or more PDEs to be solved on some or all of the input mesh. All of the PDEs in each Region are solved in a tightly coupled (i.e., single matrix) manner using one of several nonlinear solution strategies available. Users may then 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 particulars 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, an Aria 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. First, the resulting linear system is considerably smaller and contains far fewer off-diagonal contributions which can significantly increase the performance of linear solvers. Also, a resulting linear system may have a more attractive form, such as symmetric positive-definite, that 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 Sierra applications.

1.4 Constraints Equations within Aria

Aria has a unique capability associated with the specification of global constraint equations. These may be used to specify conserved quantities that are not specifically specified as part of the equations set. For example, in some electrochemistry problems where current is specified as a boundary condition, the global conservation of charge neutrality must be imposed as an additional global condition.

Constraint equations have unique issues associated with their solution.

1.5 Level Set Algorithm

Level set algorithms utilize a signed distance function F such that one material, or *phase*, is associated with regions of space where $F > 0$ and a different phase is associated with regions of space where $F < 0$. The curve or surface where $F = 0$ defines the interface between the two phases. In Goma and most other level set codes F is used to partition material property models such that the property has the appropriate values in each phase and, typically, transitions smoothly from one phase to the other. In Aria, however, F is used to partition contributions of the residual equations between the two phases.

In both cases the partitioning is done using a Heaviside function to partition the physical space into two phases which we'll label A and B. The Heaviside function $H_A(F)$ is defined such that $H_A(F) = 1$ in phase A and $H_A(F) = 0$ in phase B; in the vicinity of $F = 0$ the Heaviside function may be defined to be a smooth function that transitions from 0 to 1. Likewise, $H_B(F) = 1$ in phase B and $H_B(F) = 0$ in phase A. In fact $H_B(F)$ is defined as $H_B(F) \equiv 1 - H_A(F)$.

In Goma, this Heaviside function is used to partition the material properties such that a material property σ is defined as

$$\sigma(F, \ldots) = \sigma_A(\ldots)H_A(F) + \sigma_B(\ldots)H_B(F). \tag{1.1}$$

In Aria, however, the integrand of each residual equation is multiplied by the sum of Heaviside function so as to decompose the equation into contributions from each phase,

$$\int_V (\ldots)\ \mathrm{dV} \rightarrow \int_V H_A(F)\,(\ldots)\ \mathrm{dV} + \int_V H_B(F)\,(\ldots)\ \mathrm{dV}. \tag{1.2}$$

This formulation has a number of advantages. Material models are not functions of F so no special models need to be written and the input syntax is the same as well. Secondly, this approach is conservative for conservative governing equations. For example, the MASS and ADVECTION terms of the energy equation (see section 5.3) are proportional to ρC_p and hence, in Goma's formulation, proportional to $H^2(F)$ where as the DIFFUSION term is proportional κ and hence proportional to $H(F)$. Thus, in the vicinity of the interface $F = 0$ energy is not transported correctly between these transport modes.

In Aria, each assembly kernel has an arbitrary list of coefficients that multiply the integrand of the kernel (see section 43.13). Thus, the formulation depicted in equation 1.2 is accomplished by simply adding the appropriate Heaviside function to the list of coefficients for each kernel associated with the equation.

1.6 "Production" vs. "Experimental" Code and Known Issues

Aria is a unique application code in that it contains both production and experimental code capabilities in the same codebase. Generally speaking, well-tested capabilities are more comprehensively documented and can be considered production-ready.

Documentation of code features which possess some level of maturity but are still under development will be marked as follows with possible caveats of usage.

Beta Capability: This feature is not sufficiently tested to be classified as a full production capability.

Documentation of code features which are strictly research or "experimental" capabilities may appear in the user manual will be designated as such with a warning.

Warning: This capability is currently being researched. Use at your own risk.

Although many code features in Aria are well-tested, there may be cases where combinations of "production" features may not function correctly together. When in doubt, one may always refer to sierra-help for guidance.

A compilation of known issues are contained in release notes for each major code release which occurs roughly every six months.

1.7 Outline of the Manual

In chapter 2 we will discuss the overall environment for running Aria applications, including the layout for the Aria input deck. In chapter 5 we will present the general equations that are solved by Aria. These should be read by every user.

In later chapters, we will delve down to discuss individual line commands of the input deck. Chapter 6 discusses equation line cards (i.e., EQ), which serve to add individual equations with coupled independent unknowns to a coupled PDE representation of a region. Chapter 8 discusses how to apply initial conditions to the field variables associated with the equation sets. Chapter 9 presents the line commands associated with specifying boundary conditions. Chapter 10 introduces the concepts associated with distinguishing conditions. Chapter 11 introduces line commands associated with source terms.

Chapter 2

Getting Started

2.1 Setting The Environment-Users External to Sandia Labs

To access Sierra/Aria 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 Up Your Environment-Users at Sandia Labs

The environment for using Aria 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 Aria

This section includes some very simple examples of how to run Aria. For more information on running on some of Sandia's clusters, etc. see [3].

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

```
$  sierra -np 1 aria -i ariarun.i
```

In this example, `ariarun.i` is the Aria input file. The output – nonlinear iterations, time step information, etc. – will be written to a file called `ariarun.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 screen 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 screen):

```
$  sierra -np 1 aria -i ariarun.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 -np 1 aria -i ariarun.i -l - -a
```

Oftentimes we want to run Aria remotely or locally in a batch mode, save any standard output and perhaps even log out from a session. Unfortunately, termination of the session through either voluntary (interactive) or involuntary (timeout) log-out may in effect terminate the Aria 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 -np 1 aria -i ariarun.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 Platform-Specific Guidelines

2.4.1 ATS-1/Trinity

The ATS-1 Trinity machine has two partitions, haswell and knl, each having different Intel CPU architectures on the compute nodes. Each compute node in the haswell partition has two 16-core Intel Haswell CPUs, and each compute node in the knl partition has one 68-core Intel Knights Landing CPU. The haswell partition is supported starting with the 4.46.1 release of Sierra, the knl partition starting with the 4.46.2 release.

For both partitions we recommend a target of approximately 500,000 elements per node as a good balance between parallel efficiency and simulation turnaround time for standard thermal analysis problems. Additionally, the haswell partition is expected to be 50-75% faster than the knl partition for the 4.46 release. On the haswell partition we recommend using $32 * num\_nodes$ processors, on the KNL partition we recommend $64 * num\_nodes$. The sierra script will handle supplying the appropriate core binding flags for each partition, and the desired partition can be selected by using the "–queue-name haswell" or "–queue-name knl" option.

2.5 Aria Environment Overview

Aria is a Sierra application implementing the finite element method (FEM) for solving systems of partial differential equations (PDEs). Foremost, Aria's development targets applications which involve incompressible flow (Navier-Stokes). However, the general design of Aria lends itself to the solution of systems of PDEs describing physical processes including energy transport, species transport with reactions, electrostatics and general transport of scalar, vector and tensor quantities in two and three dimensions both transient and direct to steady state. Moreover, different regions of the physical domain (i.e., the input mesh) may have either different materials and/or different collections of physics (viz., PDEs) defined on them. These systems of equations may be solved alone, in a segregated but coupled algorithm ("loosely coupled") or as a single, fully-coupled system. Currently, Aria's loose coupling capabilities are handled by the Arpeggio application which also allows Aria to couple (loosely) to the quasistatic structural mechanics code, Adagio.

Aria's models and algorithms are integrated into the Sierra framework through the architecture illustrated in Figure 2.1. 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-proved set of procedures.

Code at the Procedure level is responsible for evolving one or more s loosely coupled sets of physics

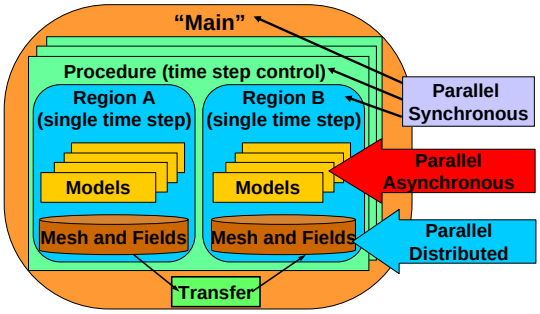


Figure 2.1. Schematic UML class diagram for the Expression subsystem.

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 Sierra Framework and Sierra Toolkit.

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.

2.6 Overview of the Input File Structure

An Aria 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. Underneath the domain is a level called the procedure, followed by the region level. Figure 1.1 shows this nesting.

The domain level contains one or more procedures. At the domain level, you 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 Aria'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 Aria'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 given a fancy name called a "solution control description". The procedure level contains a 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 tightly coupled equation system to be solved. The details include boundary conditions and initial conditions, where materials models are applied, and where surface and volumetric source terms are applied. Essentially, meshes and material properties described at the domain level are tied into the problem statement here via their names.

Global constraints equations are also specified at the region level. At the region level, specification of what gets sent to the output file and at what frequency also is made. 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 block1
End Finite Element model fluid
```

will have the same effect as

```
Begin Finite Element model fluid
  Use Material water for block1
  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 an aria region:

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

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

2.6.1 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 Aria expects to find as it parses an input file. These are *keywords, names, parameters* and *delimiters*.

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`.

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 Aria region` *name-1*

one could perhaps use

`Begin Aria region` *name_1*.

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.

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.

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 <>.

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 Aria Material Block command is

`Density` {= |IS} (R)

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

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

and

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

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

White Space

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

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.

Including Files

External text files containing input commands can be included at any point in the Aria input file using the `INCLUDEFILE` command. This command can be used in any context in the input file. To use this command, simply use the command `INCLUDEFILE` followed by the name of the file to be included. For example, the command:

```
INCLUDEFILE extrafile.i
```

would include the contents of `extrafile.i` at the locations where it is included in the input file. The included file is contained in the standard echo of the input that is provided at the beginning of the log file.

NOTE: Though this line command works well in many simple circumstances, it is known to cause issues when file names are involved. The most robust method to include files is to use Aprepro's `include` function and pre-process the input file with aprepro before running. An example is below:

```
#{include(extrafile.i)}
```

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.

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:

```
[<Operator>_]<Name>[_<Subindex>][_<Phase>][_<Component>]
```

Figure 2.2. General format of Aria's string-based naming convention for expressions. Fields in square brackets are optional.

```
Begin SIERRA Job_Indentifier
# This thermal model for Aria 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_Indentifier
```

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 -np 1 aria -check-syntax -i input.i
```

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

2.7 Fields

Fields are defined as variables which are distributed on mesh objects. For example, if the temperature is defined via Q1 interpolation on a 2D mesh consisting of quadrilaterals, then the vector of nodal temperature coefficients that make up the interpolation would be defined as the Temperature field on that mesh. Fields may be defined on any mesh object type (e.g., elements, faces, edges, nodes, node sets, and side sets), not just at nodes.

The mesh object and field data may be distributed among parallel processors via a domain decomposition algorithm. Both fields and meshes are owned at the region level. A particular field may or may not be part of Aria's solution vector for the particular region. However, all fields in Aria's solution vector are fields defined on the mesh for that region.

2.7.1 Field String-Naming Convention

Due to the dynamic nature of fields and variables in Aria a consistent naming convention must be used for sanity sake. This section describes the format of string-names of Aria Expressions. These string forms are used for input and output only; Aria has more efficient internal structures for referencing Expressions.

Briefly, the overall format is described in Figure 2.2.

Valid values of the `<Operator>` field are listed in Table 2.1. Valid values of the `<Name>` field are too numerous to list here; they include things like degrees of freedom (`VELOCITY`, `SPECIES`) and material properties

Operator	Description
(none)	"No-Op", no-operator
DT	Time derivative
GRAD	Gradient
DIV	Divergence
DET	Determinant of a 2-tensor
DETJ	Determinate of the Jacobian of transformation
SURFACE_DETJ	Determinate of the Jacobian of transformation
REF_FRAME	"No-Op" in the undeformed reference frame
GRAD_REF_FRAME	Gradient in the undeformed reference frame
DIV_REF_FRAME	Divergence in the undeformed reference frame
DETJ_REF_FRAME	Determinate of the Jacobian of transformation in the undeformed reference frame
SURFACE_DETJ_REF_FRAME	Determinate of the Jacobian of transformation in the undeformed reference frame
OLD	"No-Op" at the previous time step
GRAD_OLD	Gradient at the previous time step
DIV_OLD	Divergence at the previous time step

Table 2.1. Valid values of the <Operator> prefix.

Phase	Description
(none)	A field present in all phases within a material
A	Phase A
B	Phase B
C	Phase C

Table 2.2. Valid values of the <Phase> suffix. Phase labels are used in level set calculations only.

(VISCOSITY, ELECTRICAL_CONDUCTIVITY). The <Subindex> field can be used to designate multiple instances of a field. This is typically used for species equations. All integer values are valid subindex values but it's best to use values ≥ 1. The <Phase> field is used in level set problems. Some fields are present in "all phases" while others, such as material properties, depend on which phase is being referred to. The <Component> field allows the user to specify a particular component of vector and tensor fields; valid values are described in Table 2.3.

2.8 Equations

Equations are defined within an Aria region to represent an particular continuity equation to be solved. Within the Aria input deck, solution variables are assigned as the independent unknowns to equations. In general, there is a one-to-one correspondence between solution unknowns and equation degrees of freedom.

2.9 Equation String-Naming Convention

Similar to the field string-naming convention, equation names pose a similar requirement. This section describes the format of string-names of Aria equations. These string forms are used for input and output

Component	Description
(none)	No specified component
X	First vector component
Y	Second vector component
Z	Third vector component
XX	(1,1) 2-tensor component (default: 0)
XY	(1,2) 2-tensor component (default: 0)
XZ	(1,3) 2-tensor component (default: 0)
YX	(2,1) 2-tensor component (default: 0)
YY	(2,2) 2-tensor component (default: 0)
YZ	(2,3) 2-tensor component (default: 0)
ZX	(3,1) 2-tensor component (default: 0)
ZY	(3,2) 2-tensor component (default: 0)
ZZ	(3,3) 2-tensor component (default: 0)

Table 2.3. Valid values of the <Component> suffix. In non-Cartesian coordinate systems these may refer to, for example, radial or angular components.

String-Name	Description
TEMPERATURE	Just the temperature.
SPECIES_2	Species number two
VELOCITY_X	The first component of the velocity vector
DIV_VELOCITY	The divergence of the velocity field
DENSITY	The density
DENSITY_A	The density in level set phase A
GRAD_SPECIES_2_Y_B	The second component of the gradient of species number 2 in level set phase B

Table 2.4. Examples of well formed string names for Aria Expressions.

<Equation_Name>[_<Subindex>][_<Phase>][_<Component>]

Figure 2.3. General format of Aria's string-based naming convention for equations. Fields in square brackets are optional.

only; Aria has more efficient internal structures for referencing equations.

Briefly, the overall format is described in Figure 2.3.

Valid values of the <Equation_Name> field are numerous and changing in time. Typical values include MOMENTUM, ENERGY, SPECIES, LEVEL_SET, MESH, CURRENT and VOLTAGE; see chapter 6 for a complete description of existing equations. All integer values are valid subindex values but it's best to use values ≥ 1 – currently -1 has a special meaning of "no subindex". The <Phase> field is used in level set problems. Some fields are present in "all phases" while others, such as material properties, depend on which phase is being referred to. The <Component> field allows the user to specify a particular component of vector and tensor equation; valid values are described in Table 2.3.

2.10 Internal Field Definition

Internal to the Aria code Fields mentioned in the previous section 2.4 have alternative internal names that facilitate their algorithmic usage. In most cases this naming convention will be "usage"->field_name where "usage" categorizes its internal usage. From a user perspective the "usage" category will be "solution" and as an example the voltage solution is internally known as solution->voltage. Application output is managed independent of the application thus output requests must correspond to the internal code Field names rather than the Field name stated in the previous section.

2.11 User Fields

Situations often arise where one wishes to provide Field data storage so that data can be transferred into or out of Aria. The mechanism used for this capability is provided by the USER FIELD command line.

2.11.1 User Field

Scope:

User Field *REAL_or_INTEGER NODE_or_FACE_or_ELEMENT SCALAR_or_VECTOR_or_TENSOR Variable_name* On *Mesh_extent* [*Value* {=|are|is} *Magnitude...*]

Parameter	Value	Default
REAL_or_INTEGER	string	undefined
NODE_or_FACE_or_ELEMENT	string	undefined
SCALAR_or_VECTOR_or_TENSOR	string	undefined
Variable_name	string	undefined
Mesh_extent	string	undefined

Summary Defines a user field of standard type data_type (REAL or INTEGER) on the FEM entity (NODE, FACE, or ELEMENT) with the specified name and Sierra field type (SCALAR,

VECTOR, or TENSOR). The field will be defined on the specified mesh part or mesh group alias. Normally this field would be supplied to Aria through a transfer.

Chapter 3

Model Definition

3.1 Model Overview

The model discretization (mesh database) 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 defined within the *Finite Element Model* command block. In coupled analyses the need often arises for different discretizations of the same physical domain. In this case one must supply a different *Finite Element Model* command block for each discretization since only a single mesh can be referenced within the command block.

The input ExodusII mesh database can be constructed by a variety of mesh generators. Traditionally, portions of the mesh have been named with a convention using the meshed entity/part with a numbered index, (e.g. surface_1, block_8). Provisions are also made within Sierra to allow referencing of meshed entities that were created with specific names. From within the Aria Region command block, surface and nodeset entities/parts can be referenced by name when setting up the model but are referenced by the traditional naming convention internal to the code. Element blocks can also employ named parts in the same way as surfaces and nodesets but here the material specification for the block internal part name must be accompanied by an associated **Alias** command line within the Finite Element Model command block. This will define a mapping between the internal part name and the block name on the database. In the event that element blocks are named but the internal block name is unknown, one can employ the *io_info* Seacas tool to determine the corresponding name to internal block name mapping for all meshed element blocks.

The basic structure of an input file can be described as having three levels of commands, domain, procedure and region. The domain level is defined as the outside level commands, the procedure level is nested within the domain level and the region level is nested within the procedure. The basic structure of the input file is demonstrated on the following page.

```
Begin Sierra myJob
  .
  Begin Finite Element Model my_fem_model
   .
  End

  Begin Global Constants
   .
  End
  .
  Model definition commands
  .
  Begin Procedure My_Aria_Procedure
     .
     Begin Aria Region My_Region
      .
      use Finite Element Model my_fem_model
      .
     End
     .
  End
  .
End Sierra myJob
```

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...*

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

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

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
DatabaseName	`string`	`undefined`
InternalName	`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
Separator	`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
NewSurfaceName	`string`	`undefined`
SurfaceName	`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 Database Name

Scope: Finite Element Model

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

Parameter	Value	Default
StreamName	`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.

3.2.5 Database Type

Scope: Finite Element Model

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

Parameter	Value	Default
DatabaseTypes	`{catalyst | dof | dof_exodus | exodus | exodusii | generated | genesis | parallel_exodus}`	`undefined`

Summary The database type/format used for the mesh.

3.2.6 Global Id Mapping Backward Compatibility

Scope: Finite Element Model

Summary (Unsupported, do not use)

3.2.7 Omit Block

Scope: Finite Element Model

`Omit Block` *`BlockList...`*

Parameter	Value	Default
BlockList	`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.

3.2.8 Omit Volume

Scope: Finite Element Model

`Omit Volume` *`VolumeList...`*

Parameter	Value	Default
VolumeList	`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.

3.2.9 Time Scale Factor

Scope: Finite Element Model

`Time Scale Factor` *`Option Scale`*

Parameter	Value	Default
Scale	`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.

3.2.10 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 a 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.

3.2.11 Use Material

Scope: Finite Element Model

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

Parameter	Value	Default
MaterialName	`string`	`undefined`
VolumeList	`string...`	`undefined`

Summary Associate the given volumes with the indicated material name.

3.2.12 Decomposition Method

Scope:

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

3.2.13 Coordinate System

Scope:

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

Parameter	Value	Default
CoordinateSystem	`{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.

Beta Capability: Aria fully supports only the global Cartesian and Axisymmetric coordinate systems. Additionally cylindrical coordinates are partially supported.

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
ia_exponent	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
ia_stiffness	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
transition_jacobian	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
Mesh Entities	`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
MatName	`string`	`undefined`

Summary Associates this element block with its material properties.

3.3.7 Material =

Scope: Parameters For Block

`Material =` *`MatName`*

Parameter	Value	Default
MatName	`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
PhaseLabel	`string`	`undefined`
MaterialName	`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
ExcludeBlockList	`string...`	`undefined`

Summary List of blocks to exclude.

3.3.10 Axisymmetry Axis

Scope:

`Axisymmetry Axis {=|are|is}` *`AxisType`*

Parameter	Value	Default
AxisType	`{x|y}`	`X`

Summary When the COORDINATE SYSTEM = AXISYMMETRIC command appears within the Finite Element Model command block one can select at the Region level the coordinate axis about which the 2D axisymmetric sweep will occur. Note that there can be no overlap between the model and the axis of rotation.

3.4 Uniform Mesh Refinement

Produces initial uniform refinement for num iterations. This also precedes initialization of the Region, including setting initial conditions. This line command should be put inside the Aria Region.

Syntax:

```
INITIAL UNIFORM REFINEMENT FOR num ITERATIONS
```

3.5 Timing Overview

Overall timing information for code execution is provided by default. Alternatively, the user can increase the granularity of timing information by requesting the output explicitly using the *PRINT TIMER* command line. Here the timing information is itemized by code functionality and identify what portion of time is spent performing specific operations. Note that this command line must be applied at the Domain Scope (i.e. outside of the Procedure).

3.5.1 Print Timer Information Every

Scope:

`Print Timer Information Every` *`Procedure-step-interval`* `Steps` *`Checkpointed`*

Parameter	Value	Default
Procedure-step-interval	`integer`	`undefined`
Checkpointed	`{accumulated|checkpointed}`	`undefined`

Summary Specifies the procedure step count interval to print timer information

3.6 Global Constants

Scope: Sierra

```
Begin Global Constants empty
    Gravity Vector {= | are | is} Gravity_1 Gravity_2 Gravity_3
    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.

3.6.1 Gravity Vector

Scope: Global Constants

```
Gravity Vector {= | are | is} Gravity_1 Gravity_2 Gravity_3
```

Parameter	Value	Default
Gravity	real_1 real_2 real_3	undefined

Summary Gravity constant in vector form, acceleration components.

3.6.2 Ideal Gas Constant

Scope: Global Constants

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

Parameter	Value	Default
Sigma	real	undefined

Summary Ideal gas constant. extbfNOTE: 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.

3.6.3 K-E Turbulence Model Parameter

Scope: Global Constants

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

Parameter	Value	Default
Param	string	undefined
Value	real	undefined

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

3.6.4 K-W Turbulence Model Parameter

Scope: Global Constants

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

Parameter	Value	Default
Param	`string`	`undefined`
Value	`real`	`undefined`

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

3.6.5 Les Turbulence Model Parameter

Scope: Global Constants

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

Parameter	Value	Default
Param	`string`	`undefined`
Value	`real`	`undefined`

Summary LES turbulence model parameters.

3.6.6 Light Speed

Scope: Global Constants

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

Parameter	Value	Default
LightSpeed	`real`	`undefined`

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

3.6.7 Planck Constant

Scope: Global Constants

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

Parameter	Value	Default
PlanckConstant	`real`	`undefined`

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

3.6.8 Stefan Boltzmann Constant

Scope: Global Constants

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

Parameter	Value	Default
Sigma	`real`	`undefined`

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

3.6.9 Turbulence Model

Scope: Global Constants

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

Parameter	Value	Default
Param	`string`	`undefined`
Value	`real`	`undefined`

Summary Turbulence model Schmidt and Prandtl numbers

3.7 Function Overview

User supplied functions are supported by Aria. The functions can be either tabular or analytical, where the most prevalent use of this capability is tabular data functions. These functions can in turn be tied to different features and models in the applications. Examples of this usage would be to define material data or load profiles, either spatially or in time.

3.8 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` $Titles_1$ $Titles_2$`...`

`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.

3.8.1 Abscissa

Scope: Definition For Function

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

Parameter	Value	Default
Name	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 scaled_abscissa = scale * (abscissa + abscissa_offset).

3.8.2 Abscissa Offset

Scope: Definition For Function

```
Abscissa Offset {=|are|is} Abscissa_offset
```

Parameter	Value	Default
Abscissa_offset	real	undefined

Summary Alias for X OFFSET

3.8.3 Abscissa Scale

Scope: Definition For Function

`Abscissa Scale {=|are|is}` *`Abscissa_scale`*

Parameter	Value	Default
Abscissa_scale	`real`	`undefined`

Summary Alias for X SCALE

3.8.4 At Discontinuity Evaluate To

Scope: Definition For Function

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.

3.8.5 Column Titles

Scope: Definition For Function

`Column Titles` *`Titles`*$_1$ *`Titles`*$_2$`...`

Parameter	Value	Default
Titles	`string_1 string_2...`	`undefined`

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

3.8.6 Data File

Scope: Definition For Function

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

Parameter	Value	Default
filename	`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 by in tabular columns seperated by whitespace or commas.

3.8.7 Debug

Scope: Definition For Function

Summary Prints functions to the log file.

3.8.8 Differentiate Expression

Scope: Definition For Function

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

Parameter	Value	Default
Expr	(expression)	undefined

Summary Specifies the expression of derivative of evaluation expression.

3.8.9 Evaluate Expression

Scope: Definition For Function

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

Parameter	Value	Default
Expr	(expression)	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) maginitude 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

3.8.10 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.

3.8.11 Expression Variable:

Scope: Definition For Function

`Expression Variable:` ***`Expr = VarType value_var_name...`***

Parameter	Value	Default
Expr	`string`	`undefined`
value_var_name	`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

3.8.12 Expression Variable:

Scope: Definition For Function

`Expression Variable:` ***`Expr`***

Parameter	Value	Default
Expr	`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 - 0.0) + ((my-0.25)/0.5)*stretchxy)" END

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

3.8.13 Ordinate

Scope: Definition For Function

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

Parameter	Value	Default
Name	`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).

3.8.14 Ordinate Offset

Scope: Definition For Function

`Ordinate Offset {=|are|is}` *`Ordinate_offset`*

Parameter	Value	Default
Ordinate_offset	`real`	`undefined`

Summary Alias for Y OFFSET

3.8.15 Ordinate Scale

Scope: Definition For Function

`Ordinate Scale {=|are|is}` *`Ordinate_scale`*

Parameter	Value	Default
Ordinate_scale	`real`	`undefined`

Summary Alias for Y SCALE

3.8.16 Scale By

Scope: Definition For Function

`Scale By` *`x`*

Parameter	Value	Default
x	`real`	`undefined`

Summary Specifies a scale factor to be applied.

3.8.17 Type

Scope: Definition For Function

Summary Specifies the type of function.

3.8.18 X Offset

Scope: Definition For Function

`X Offset {=|are|is}` *`X_offset`*

Parameter	Value	Default
X_offset	`real`	`undefined`

Summary Sets an offset for the x-axis

3.8.19 X Scale

Scope: Definition For Function

```
X Scale {=|are|is} X_scale
```

Parameter	Value	Default
X_scale	`real`	`undefined`

Summary Sets a scale factor for the x-axis

3.8.20 Y Offset

Scope: Definition For Function

```
Y Offset {=|are|is} Y_offset
```

Parameter	Value	Default
Y_offset	`real`	`undefined`

Summary Sets an offset for the y-axis

3.8.21 Y Scale

Scope: Definition For Function

```
Y Scale {=|are|is} Y_scale
```

Parameter	Value	Default
Y_scale	`real`	`undefined`

Summary Sets a scale factor for the y-axis

3.9 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.

3.9.1

Scope: Values

Xyvalues...

Parameter	Value	Default
Xyvalues	`real...`	`undefined`

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

3.10 Restarting

Convenient utilities are provided 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 25.6. Continuation of a job using restart data output is invoked using the command line which follows.

Often one wishes to restart an analysis by initializing with results from another analysis. For this case Aria provides another means, IC READ_FILE 8.7, by which to begin the job with variables from existing results.

Chapter 4

Material Properties

4.1 Aria Material Overview

Material property values are prescribed for unique material names using input line commands within an "Aria Material" command block. The association of these material properties with mesh entities is specified within the Finite Element Model command block 3.1.

The general format of the material property line commands as follows

`Property Type = Model_Name Value_Name = Value Model_Parameter1 ....  Model_ParameterN.`

For some material properties, clarification of how the property is to be used must be also be supplied. This clarification usually impacts the computational efficiency of how the property values are being used. As an example, in the case of diffusion coefficients one needs to specify whether it will be used in a scalar or tensor based formulation of flux.

Chemical material models solved using the CHEMEQ library require an entire set of parameters. In this case those parameters are defined within a `Parameters for CHEMEQ` command block embedded within the Aria Material command block. A description of the CHEMEQ related parameters is contained in a subsequent chapter 36.2.

The material models `Encore Function`, `Global`, `User Function`, `Polynomial`, `User Field` and `Exponential` are termed Generic and are available for any material property evaluation. The `Constant` material model is also available for all properties but differs syntactically for each material property.

Known Issue: The feature toggling capability is not currently supported for setting material property values, although the line command to do so is listed when the material is specified as a `User_Function`.

Aria material command blocks appear in the Domain scope of the input file as illustrated below.

```
Begin Sierra myJob
  .
  Begin Aria Material my_mat
    .
    material property commands
    .
  End   Aria Material my_mat
  .
  Begin Procedure My_Aria_Procedure
     .
     Begin Aria Region My_Region
```

```
      .
      .
     End   Aria Region My_Region
      .
  End   Procedure My_Aria_Procedure
  .
End   Sierra myJob
```

4.1.1 Generic: Encore Function

Scope: Aria Material

```
   Generic:  Encore Function
<Material Property> [ {of|species|subindex} Species  |{in|material_phase} MaterialPhase ]{@
|at|for|in|on|over} Mesh Extent Name = Encore_Function [ Using Data Specification Data Spec
Name ][ Name = name  |Result_Name = result_name  |Eval_Type = eval_type ]
```

Parameter	Value	Default
<Material Property>	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
result_name	"string"	undefined
eval_type	"string"	undefined

Summary Value from an Encore function

4.1.2 Generic: Global

Scope: Aria Material

```
   Generic:  Global
<Material Property> [ {of|species|subindex} Species  |{in|material_phase} MaterialPhase ]{@
|at|for|in|on|over} Mesh Extent Name = Global [ Using Data Specification Data Spec Name ][
Global_Name = global_name ]
```

Parameter	Value	Default
<Material Property>	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
global_name	"string"	undefined

Summary Value from a global variable

4.1.3 Generic: User Function

Scope: Aria Material

Generic: User Function
<Material Property> [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* = User_Function [Using Data Specification *Data Spec Name*][Name = *name* |X = *x* |X_Multiplier = *x_multiplier* |Multiplier = *multiplier* |Toggle = *toggle*]

Parameter	Value	Default
<Material Property>	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
x	"string"	undefined
x_multiplier	real	undefined
multiplier	real	undefined
toggle	"string"	undefined

Summary Value from a user function

4.1.4 Generic: Polynomial

Scope: Aria Material

Generic: Polynomial
<Material Property> [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* = Polynomial [Using Data Specification *Data Spec Name*][Variable = *variable* |Order = *order* |Variable_Offset = *variable_offset* |C0 = *c0* |C1 = *c1* |C2 = *c2* |C3 = *c3* |C4 = *c4* |C5 = *c5* |C6 = *c6* |C7 = *c7* |C8 = *c8*]

Parameter	Value	Default
<Material Property>	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
variable	"string"	undefined
order	integer	undefined
variable_offset	real	undefined
c0	"string"	undefined
c1	real	undefined
c2	real	undefined
c3	real	undefined
c4	real	undefined
c5	real	undefined
c6	real	undefined
c7	real	undefined
c8	real	undefined

Summary Value from a polynomial function

4.1.5 Generic: User Field

Scope: Aria Material

Generic: User Field
<Material Property> [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* = User_Field [Using Data Specification *Data Spec Name*][Name = *name* | Scaling = *scaling* | Global_Var = *global_var*]

Parameter	Value	Default
<Material Property>	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
scaling	real	undefined
global_var	"string"	undefined

Summary Value from a user field

4.1.6 Generic: Exponential

Scope: Aria Material

Generic: Exponential
<Material Property> [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* = Exponential [Using Data Specification *Data Spec Name*][Variable = *variable* | Constant = *constant* | Multiplier = *multiplier* | Exponent = *exponent*]

Parameter	Value	Default
<Material Property>	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
variable	"string"	undefined
constant	real	undefined
multiplier	real	undefined
exponent	real	undefined

Summary Value from an exponential function

4.1.7 Absorption Coefficient

Scope: Aria Material

Absorption Coefficient [{of | species | subindex} *Species*]= Constant [K = *k*]

Parameter	Value	Default
Species	string	undefined
k	real	undefined

Summary Constant value

4.1.8 Absorption Coefficient: Copied

Scope: Aria Material

`Absorption Coefficient: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.9 Absorption Coefficient: User Plugin

Scope: Aria Material

`Absorption Coefficient: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.10 Absorption Coefficient: Porous Phase Average

Scope: Aria Material

`Absorption Coefficient: Porous Phase Average [ {of | species | subindex} Species ]= Porous_Phase_Averag [ Liquid_Phase_Name = Liquid_Phase_Name |Gas_Phase_Name = Gas_Phase_Name ]`

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined

Summary Absorption coefficient for a multiphase material of two or three phases

4.1.11 Absorption Cross Section

Scope: Aria Material

`Absorption Cross Section [ {of | species | subindex} Species ]= Constant [ Abs = abs ]`

Parameter	Value	Default
Species	string	undefined
abs	real	undefined

Summary Constant value

4.1.12 Absorption Cross Section: Copied

Scope: Aria Material

Absorption Cross Section: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.13 Absorption Cross Section: User Plugin

Scope: Aria Material

Absorption Cross Section: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.14 Absorption Cross Section: Linearized

Scope: Aria Material

Absorption Cross Section: Linearized [{of|species|subindex} *Species*]= Linearized [Sigma_0 = *sigma_0* |D_Sigma_D_Rho = *d_sigma_d_rho* |D_Sigma_D_T = *d_sigma_d_t* |T_Sigma_0 = *t_sigma_0* |Rho_Sigma_0 = *rho_sigma_0*]

Parameter	Value	Default
Species	string	undefined
sigma_0	real	undefined
d_sigma_d_rho	real	undefined
d_sigma_d_t	real	undefined
t_sigma_0	real	undefined
rho_sigma_0	real	undefined

Summary Linearized absorption cross section

4.1.15 Absorption Cross Section: Calore User Sub

Scope: Aria Material

Absorption Cross Section: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Calore user subroutine absorption cross section

4.1.16 Advected Bubble

Scope: Aria Material

Advected Bubble [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.17 Advected Bubble: Copied

Scope: Aria Material

Advected Bubble: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.18 Advected Bubble: User Plugin

Scope: Aria Material

Advected Bubble: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.19 Advection Velocity

Scope: Aria Material

Advection Velocity [{of | species | subindex} *Species*]= Constant [Valuex = *valuex* | Valuey = *valuey* | Valuez = *valuez*]

Parameter	Value	Default
Species	string	undefined
valuex	real	undefined
valuey	real	undefined
valuez	real	undefined

Summary Constant value

4.1.20 Advection Velocity: User Plugin

Scope: Aria Material

Advection Velocity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.21 Air Water Mass Flux: Fickian

Scope: Aria Material

Air Water Mass Flux: Fickian [{of | species | subindex} *Species*]= Fickian [ ]

Parameter	Value	Default
Species	string	undefined

Summary Fickian diffusion for the air-water mass flux

4.1.22 Altitude

Scope: Aria Material

`Altitude [ {of|species|subindex} Species ]= Constant [ A = a ]`

Parameter	Value	Default
Species	string	undefined
a	real	undefined

Summary Constant value

4.1.23 Altitude: Copied

Scope: Aria Material

`Altitude: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.24 Altitude: User Plugin

Scope: Aria Material

`Altitude: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.25 Ambient Pressure

Scope: Aria Material

`Ambient Pressure [ {of|species|subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.26 Ambient Pressure: Copied

Scope: Aria Material

Ambient Pressure: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.27 Ambient Pressure: User Plugin

Scope: Aria Material

Ambient Pressure: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.28 Annulus Diameter Ratio

Scope: Aria Material

Annulus Diameter Ratio [{of | species | subindex} *Species*]= Constant [R = *r*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined

Summary Constant value

4.1.29 Annulus Diameter Ratio: Copied

Scope: Aria Material

Annulus Diameter Ratio: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.30 Annulus Diameter Ratio: User Plugin

Scope: Aria Material

Annulus Diameter Ratio: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.31 Annulus Diameter Ratio: Spatial User Function

Scope: Aria Material

Annulus Diameter Ratio: Spatial User Function [{of|species|subindex} *Species*]= Spatial_User_Func
[Threshold = *threshold* |Multiplier = *multiplier* |Name = *name* |Bar = *bar* |X = *x*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined

Summary Spatially varying annulus diameter ratio based on a coordinate dependent user function

4.1.32 Annulus Diameter Ratio: Correlation

Scope: Aria Material

Annulus Diameter Ratio: Correlation [{of|species|subindex} *Species*]= Correlation [R = *r* |Threshold = *threshold* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
threshold	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation annulus diameter ratio based on hydraulic diameter and wetted perimeter

4.1.33 Annulus Diameter Ratio: Correlation Spatial User Function

Scope: Aria Material

Annulus Diameter Ratio: Correlation Spatial User Function [{of|species|subindex} *Species*]= Correlation_Spatial_User_Function [Threshold = *threshold* |Multiplier = *multiplier* |Name = *name* |Bar = *bar* |X = *x* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined
bulk_node_interface	"string"	undefined

Summary Correlation spatially varying annulus diameter ratio based on a coordinate dependent user function

4.1.34 Auxiliary Mesh Field: Scalar

Scope: Aria Material

Auxiliary Mesh Field: Scalar [{of | species | subindex} *Species*]= Scalar [Field_Name = *field_name* | Expression_Name = *expression_name*]

Parameter	Value	Default
Species	string	undefined
field_name	"string"	undefined
expression_name	"string"	undefined

Summary Scalar auxiliary mesh field

4.1.35 Bar Area

Scope: Aria Material

Bar Area [{of | species | subindex} *Species*]= Constant [A = *a*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined

Summary Constant value

4.1.36 Bar Area: Copied

Scope: Aria Material

Bar Area: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.37 Bar Area: User Plugin

Scope: Aria Material

`Bar Area: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.38 Bar Area: Element Attribute

Scope: Aria Material

`Bar Area: Element Attribute [ {of|species|subindex} Species ]= Element_Attribute [ Multiplier = multiplier |Attribute_Name = attribute_name ]`

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
attribute_name	"string"	undefined

Summary Distribution Factor for the bar area

4.1.39 Bar Perimeter

Scope: Aria Material

`Bar Perimeter [ {of|species|subindex} Species ]= Constant [ P = p ]`

Parameter	Value	Default
Species	string	undefined
p	real	undefined

Summary Constant value

4.1.40 Bar Perimeter: Copied

Scope: Aria Material

`Bar Perimeter: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.41 Bar Perimeter: User Plugin

Scope: Aria Material

Bar Perimeter: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.42 Bar Perimeter: Element Attribute

Scope: Aria Material

Bar Perimeter: Element Attribute [{of | species | subindex} *Species*]= Element_Attribute [Multiplier = *multiplier* | Attribute_Name = *attribute_name*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
attribute_name	"string"	undefined

Summary Distribution Factor for the bar perimeter

4.1.43 Bc Rad Reference Temperature

Scope: Aria Material

Bc Rad Reference Temperature [{of | species | subindex} *Species*]= Constant [T_Ref = *t_ref*]

Parameter	Value	Default
Species	string	undefined
t_ref	real	undefined

Summary Constant value

4.1.44 Bc Rad Reference Temperature: Copied

Scope: Aria Material

Bc Rad Reference Temperature: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.45 Bc Rad Reference Temperature: User Plugin

Scope: Aria Material

Bc Rad Reference Temperature: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.46 Bc Rad Reference Temperature: Bulk Node

Scope: Aria Material

Bc Rad Reference Temperature: Bulk Node [{of|species|subindex} *Species*]= Bulk_Node [Name = *name*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined

Summary Use a bulk node temperature as the reference temperature for radiation boundary conditions

4.1.47 Bc Rad Reference Temperature: Calore User Sub

Scope: Aria Material

Bc Rad Reference Temperature: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.48 Bc Rad Reference Temperature: Fortran

Scope: Aria Material

Bc Rad Reference Temperature: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine reference temperature.

4.1.49 Bc Reference Temperature

Scope: Aria Material

```
Bc Reference Temperature [ {of|species|subindex} Species ]= Constant [ T_Ref = t_ref ]
```

Parameter	Value	Default
Species	string	undefined
t_ref	real	undefined

Summary Constant value

4.1.50 Bc Reference Temperature: Copied

Scope: Aria Material

```
Bc Reference Temperature: Copied [ {of|species|subindex} Species ]= Copied [ Source =
source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.51 Bc Reference Temperature: User Plugin

Scope: Aria Material

```
Bc Reference Temperature: User Plugin [ {of|species|subindex} Species ]= User_Plugin [
Name = Name |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.52 Bc Reference Temperature: Bulk Node

Scope: Aria Material

Bc Reference Temperature: Bulk Node [{of | species | subindex} *Species*]= Bulk_Node [Name = *name*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined

Summary Use a bulk node temperature as the reference temperature for convection boundary conditions

4.1.53 Bc Reference Temperature: Calore User Sub

Scope: Aria Material

Bc Reference Temperature: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Name = *name* | Type = *type* | Multiplier = *multiplier* | Material_Data_Block = *material_data_block* | Data = *data* | Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.54 Bc Reference Temperature: Computed Adiabatic Wall

Scope: Aria Material

Bc Reference Temperature: Computed Adiabatic Wall [{of | species | subindex} *Species*]= Computed_Adial [T_On = *t_on* | T_Off = *t_off* | Specific_Heat_Function = *specific_heat_function* | Conductivity_Function = *conductivity_function* | Viscosity_Function = *viscosity_function* | Offset_Direction = *offset_direction* | Coord_Offset = *coord_offset* | Gas_Constant = *gas_constant*]

Parameter	Value	Default
Species	string	undefined
t_on	real	undefined
t_off	real	undefined
specific_heat_function	"string"	undefined
conductivity_function	"string"	undefined
viscosity_function	"string"	undefined
offset_direction	"string"	undefined
coord_offset	"string"	undefined
gas_constant	real	undefined

Summary Adiabatic wall temperature computed based upon freestream and recovery factor.

4.1.55 Bc Reference Temperature: Fortran

Scope: Aria Material

Bc Reference Temperature: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine reference temperature.

4.1.56 Beta

Scope: Aria Material

Beta [{of|species|subindex} *Species*]= Constant [Beta = *beta*]

Parameter	Value	Default
Species	string	undefined
beta	real	undefined

Summary Constant value

4.1.57 Beta: Copied

Scope: Aria Material

Beta: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.58 Beta: User Plugin

Scope: Aria Material

Beta: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_st*
]

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_parameter	`"string"`	`undefined`

Summary Value from a user plugin

4.1.59 Beta: Converted

Scope: Aria Material

`Beta: Converted [ {of|species|subindex} Species ]= Converted [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Coefficient of thermal stress computed from Poisson's ratio, thermal expansion coefficient and elastic modulus

4.1.60 Beta: Linear

Scope: Aria Material

`Beta: Linear [ {of|species|subindex} Species ]= Linear [ A = a |B = b ]`

Parameter	Value	Default
Species	`string`	`undefined`
a	`real`	`undefined`
b	`real`	`undefined`

Summary Coefficient of thermal stress as a linear function of temperature

4.1.61 Body Acceleration

Scope: Aria Material

`Body Acceleration [ {of|species|subindex} Species ]= Constant [ Valuex = valuex |Valuey = valuey |Valuez = valuez ]`

Parameter	Value	Default
Species	`string`	`undefined`
valuex	`real`	`undefined`
valuey	`real`	`undefined`
valuez	`real`	`undefined`

Summary Constant value

4.1.62 Body Acceleration: User Plugin

Scope: Aria Material

Body Acceleration: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.63 Boundary Entrained Enthalpy

Scope: Aria Material

Boundary Entrained Enthalpy [{of | species | subindex} *Species*]= Constant [H = *h*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.64 Boundary Entrained Enthalpy: Copied

Scope: Aria Material

Boundary Entrained Enthalpy: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.65 Boundary Entrained Enthalpy: User Plugin

Scope: Aria Material

Boundary Entrained Enthalpy: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.66 Boundary Entrained Mass Fraction

Scope: Aria Material

`Boundary Entrained Mass Fraction [ {of | species | subindex} Species ]= Constant [ Y = y ]`

Parameter	Value	Default
Species	string	undefined
y	real	undefined

Summary Constant value

4.1.67 Boundary Entrained Mass Fraction: Copied

Scope: Aria Material

`Boundary Entrained Mass Fraction: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.68 Boundary Entrained Mass Fraction: User Plugin

Scope: Aria Material

`Boundary Entrained Mass Fraction: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.69 Boundary Pressure

Scope: Aria Material

`Boundary Pressure [ {of | species | subindex} Species ]= Constant [ P = p ]`

Parameter	Value	Default
Species	string	undefined
p	real	undefined

Summary Constant value

4.1.70 Boundary Pressure: Copied

Scope: Aria Material

`Boundary Pressure: Copied [ {of|species|subindex} `*`Species`*` ]= Copied [ Source = `*`source`*` ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.71 Boundary Pressure: User Plugin

Scope: Aria Material

`Boundary Pressure: User Plugin [ {of|species|subindex} `*`Species`*` ]= User_Plugin [ Name = `*`Name`*` |`*`magic_trailing_string_vector_param`*` ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.72 Bulk Conductivity

Scope: Aria Material

`Bulk Conductivity [ {of|species|subindex} `*`Species`*` ]= Constant [ Kbulk = `*`Kbulk`*` ]`

Parameter	Value	Default
Species	string	undefined
Kbulk	real	undefined

Summary Constant value

4.1.73 Bulk Conductivity: Copied

Scope: Aria Material

`Bulk Conductivity: Copied [ {of|species|subindex} `*`Species`*` ]= Copied [ Source = `*`source`*` ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.74 Bulk Conductivity: User Plugin

Scope: Aria Material

`Bulk Conductivity: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.75 Bulk Conductivity: T Exponent

Scope: Aria Material

`Bulk Conductivity: T Exponent [ {of | species | subindex} Species ]= T_Exponent [ Kbulk_Ref = kbulk_ref | T_Ref = t_ref | N = n ]`

Parameter	Value	Default
Species	`string`	`undefined`
kbulk_ref	`real`	`undefined`
t_ref	`real`	`undefined`
n	`real`	`undefined`

Summary Thermal conductivity for bulk material.$\kappa_{\text{bulk}} = \kappa_{\text{ref}} \frac{\rho_{\text{solid}}}{\rho_{\text{bulk}}} \left(\frac{T}{T_{\text{ref}}} \right)^n$.

4.1.76 Bulk Conductivity: Volume Average

Scope: Aria Material

`Bulk Conductivity: Volume Average [ {of | species | subindex} Species ]= Volume_Average [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Bulk conductivity computed from volume average of species bulk conductivities

4.1.77 Bulk Conductivity: Linear Temperature And Density

Scope: Aria Material

`Bulk Conductivity: Linear Temperature And Density [ {of | species | subindex} Species ]= Linear_Tempera [ C_0 = c_0 | C_Rho = C_rho | C_T = c_t ]`

Parameter	Value	Default
Species	string	undefined
c_0	real	undefined
C_rho	real	undefined
c_t	real	undefined

Summary Bulk conductivity that is a linear function of both temperature and density. $\kappa_{bulk} = C_0 + C_\rho \rho + C_T T$

4.1.78 Bulk Conductivity: Porous Phase Average

Scope: Aria Material

`Bulk Conductivity: Porous Phase Average [ {of|species|subindex} `*`Species`*` ]= Porous_Phase_Average [ Liquid_Phase_Name = `*`Liquid_Phase_Name`*` |Gas_Phase_Name = `*`Gas_Phase_Name`*` ]`

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined

Summary Bulk conductivity for a multiphase material of two or three phases

4.1.79 Bulk Density

Scope: Aria Material

`Bulk Density [ {of|species|subindex} `*`Species`*` ]= Constant [ Rho_B = `*`rho_b`*` ]`

Parameter	Value	Default
Species	string	undefined
rho_b	real	undefined

Summary Constant value

4.1.80 Bulk Density: Copied

Scope: Aria Material

`Bulk Density: Copied [ {of|species|subindex} `*`Species`*` ]= Copied [ Source = `*`source`*` ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.81 Bulk Density: User Plugin

Scope: Aria Material

Bulk Density: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.82 Bulk Density: T Exponent

Scope: Aria Material

Bulk Density: T Exponent [{of | species | subindex} *Species*]= T_Exponent [Rho_Ref = *rho_ref* | T_Ref = *t_ref* | N = *n*]

Parameter	Value	Default
Species	string	undefined
rho_ref	real	undefined
t_ref	real	undefined
n	real	undefined

Summary Bulk density as a power of normalized temperature$\rho_{\text{bulk}} = \rho_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^n$.

4.1.83 Bulk Density: Single Component Ideal Gas

Scope: Aria Material

Bulk Density: Single Component Ideal Gas [{of | species | subindex} *Species*]= Single_Component_Ideal_
[P_Ref = *p_ref* | T_Ref = *t_ref* | M = *m* | R = *r*]

Parameter	Value	Default
Species	string	undefined
p_ref	real	undefined
t_ref	real	undefined
m	real	undefined
r	real	undefined

4.1.84 Bulk Element Pressure

Scope: Aria Material

Bulk Element Pressure [{of | species | subindex} *Species*]= Constant [V = *v* | Bulk_Node = *bulk_node*]

Parameter	Value	Default
Species	string	undefined
v	"string"	undefined
bulk_node	"string"	undefined

Summary Constant value for the bulk element pressure

4.1.85 Bulk Element Pressure: Integrated

Scope: Aria Material

`Bulk Element Pressure: Integrated [ {of|species|subindex} Species ]= Integrated [ Name = name ]`

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined

Summary Bulk element pressure computed from equation of state

4.1.86 Bulk Element Pressure: Calore User Sub

Scope: Aria Material

`Bulk Element Pressure: Calore User Sub [ {of|species|subindex} Species ]= Calore_User_Sub [ Bulk_Node = bulk_node ]`

Parameter	Value	Default
Species	string	undefined
bulk_node	"string"	undefined

Summary Bulk element pressure from Calore User Subroutine

4.1.87 Bulk Element Volume

Scope: Aria Material

`Bulk Element Volume [ {of|species|subindex} Species ]= Constant [ V = v |Bulk_Node = bulk_node ]`

Parameter	Value	Default
Species	string	undefined
v	"string"	undefined
bulk_node	"string"	undefined

Summary Constant value for the bulk element volume

4.1.88 Bulk Element Volume: Integrated

Scope: Aria Material

`Bulk Element Volume: Integrated [ {of|species|subindex} Species ]= Integrated [ Name = name ]`

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined

Summary Bulk element volume computed from bounding surfaces

4.1.89 Bulk Element Volume: Calore User Sub

Scope: Aria Material

`Bulk Element Volume: Calore User Sub [ {of|species|subindex} Species ]= Calore_User_Sub [ Bulk_Node = bulk_node ]`

Parameter	Value	Default
Species	`string`	`undefined`
bulk_node	`"string"`	`undefined`

Summary Bulk element volume from Calore User Subroutine

4.1.90 Bulk Mass Density: Porous Density

Scope: Aria Material

`Bulk Mass Density: Porous Density [ {of|species|subindex} Species ]= Porous_Density [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Bulk mass density computed from density and porosity.

4.1.91 Bulk Mass Density: Density

Scope: Aria Material

`Bulk Mass Density: Density [ {of|species|subindex} Species ]= Density [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Bulk mass density copied from density.

4.1.92 Bulk Mass Density: Mass Fraction Density

Scope: Aria Material

`Bulk Mass Density: Mass Fraction Density [ {of|species|subindex} Species ]= Mass_Fraction_Density [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Bulk mass density computed from mass fraction and density.

4.1.93 Bulk Mass Density: Mass Fraction Porous Density

Scope: Aria Material

Bulk Mass Density: Mass Fraction Porous Density [{of | species | subindex} *Species*]= Mass_Fraction_Po
[ ]

Parameter	Value	Default
Species	string	undefined

Summary Bulk mass density computed from mass fraction, porosity and density.

4.1.94 Bulk Mass Density: Multiphase Porous Density

Scope: Aria Material

Bulk Mass Density: Multiphase Porous Density [{of | species | subindex} *Species*]= Multiphase_Porous_
[ ]

Parameter	Value	Default
Species	string	undefined

Summary Bulk mass density computed from density, saturation and porosity for a multiphase material.

4.1.95 Bulk Mass Density

Scope: Aria Material

Bulk Mass Density [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.96 Bulk Mass Density: Copied

Scope: Aria Material

Bulk Mass Density: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*
]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.97 Bulk Mass Density: User Plugin

Scope: Aria Material

Bulk Mass Density: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name =
Name | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.98 Bulk Viscosity

Scope: Aria Material

```
Bulk Viscosity [ {of | species | subindex} Species ]= Constant [ Kappa = kappa ]
```

Parameter	Value	Default
Species	string	undefined
kappa	real	undefined

Summary Constant value

4.1.99 Bulk Viscosity: Copied

Scope: Aria Material

```
Bulk Viscosity:  Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.100 Bulk Viscosity: User Plugin

Scope: Aria Material

```
Bulk Viscosity:  User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name
| magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.101 Bulk Viscosity: Curing Foam

Scope: Aria Material

```
Bulk Viscosity:  Curing Foam [ {of | species | subindex} Species ]= Curing_Foam [ Extent_Subindex
= extent_subindex  | Extent_Species_Name = extent_species_name  | Extent_Species_Phase = extent_species_ph
```

| Mu_A = *mu_a* | T_Mu = *T_mu* | N_Mu = *n_mu* | Ksi_C = *ksi_c* | M_Ksi_C = *m_ksi_c* | E_Mu = *E_mu* | R = *r*]

Parameter	Value	Default
Species	string	undefined
extent_subindex	integer	undefined
extent_species_name	"string"	undefined
extent_species_phase	"string"	undefined
mu_a	real	undefined
T_mu	real	undefined
n_mu	real	undefined
ksi_c	real	undefined
m_ksi_c	real	undefined
E_mu	real	undefined
r	real	undefined

4.1.102 Bulk Viscosity: Phase Average

Scope: Aria Material

Bulk Viscosity: Phase Average [{of | species | subindex} *Species*]= Phase_Average [Subindex_A = *subindex_a* | Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.103 Bulk Viscosity: Interpolated Phase Average

Scope: Aria Material

Bulk Viscosity: Interpolated Phase Average [{of | species | subindex} *Species*]= Interpolated_Phase_A
[Subindex_A = *subindex_a* | Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.104 Burn Front Width

Scope: Aria Material

Burn Front Width [{of | species | subindex} *Species*]= Constant [Width = *width* | Use_Crossing_Time = *use_crossing_time*]

Parameter	Value	Default
Species	string	undefined
width	real	undefined
use_crossing_time	integer	undefined

Summary Specification of a constant width burn front

4.1.105 Burn Heat Release

Scope: Aria Material

```
Burn Heat Release [ {of | species | subindex} Species ]= Constant [ Value = value ]
```

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.106 Burn Heat Release: Copied

Scope: Aria Material

```
Burn Heat Release:  Copied [ {of | species | subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.107 Burn Heat Release: User Plugin

Scope: Aria Material

```
Burn Heat Release:  User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name =
Name | magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.108 Burn Initiation: Temperature

Scope: Aria Material

Burn Initiation: Temperature [{of | species | subindex} *Species*]= Temperature [Tign = *Tign* | Min_Ignition_Spacing = *min_ignition_spacing*]

Parameter	Value	Default
Species	string	undefined
Tign	real	undefined
min_ignition_spacing	real	undefined

Summary Thermally activated burn front. If min_ignition_spacing is provided then multiple burn initiations are supported, otherwise only one initiation per block is supported. Pick a minimum ignition spacing (d) that is several times larger than your interface width ($d > 3w$), and that provides $Da >> 10$, where $Da = \frac{v_b d}{\alpha}$ (α = thermal diffusivity and v_b = burn speed)

4.1.109 Burn Speed

Scope: Aria Material

Burn Speed [{of | species | subindex} *Species*]= Constant [Value = *Value*]

Parameter	Value	Default
Species	string	undefined
Value	real	undefined

Summary Constant value

4.1.110 Capillary Pressure

Scope: Aria Material

Capillary Pressure [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.111 Capillary Pressure: Copied

Scope: Aria Material

Capillary Pressure: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.112 Capillary Pressure: User Plugin

Scope: Aria Material

Capillary Pressure: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.113 Capillary Pressure: Van Genuchten

Scope: Aria Material

Capillary Pressure: Van Genuchten [{of | species | subindex} *Species*]= Van_Genuchten [Residual_Wetti = *residual_wetting_phase_saturation* | Residual_Nonwetting_Phase_Saturation = *residual_nonwetting_phase_s* | Beta_Field = *beta_field* | Pcref_Field = *pcref_field* | Sgr_Field = *sgr_field* | Slr_Field = *slr_field* | Reference_Capillary_Pressure = *reference_capillary_pressure* | Beta = *beta* | Wetting_Phase = *wetting_phase*]

Parameter	Value	Default
Species	string	undefined
residual_wetting_phase_saturation	real	undefined
residual_nonwetting_phase_saturation	real	undefined
beta_field	"string"	undefined
pcref_field	"string"	undefined
sgr_field	"string"	undefined
slr_field	"string"	undefined
reference_capillary_pressure	real	undefined
beta	real	undefined
wetting_phase	"string"	undefined

Summary A Van Genuchten model for air/water capillary pressure

4.1.114 Capillary Pressure: Encore Saturation

Scope: Aria Material

Capillary Pressure: Encore Saturation [{of | species | subindex} *Species*]= Encore_Saturation [Name = *name* | Result_Name = *result_name* | Eval_Type = *eval_type*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
result_name	"string"	undefined
eval_type	"string"	undefined

Summary A Van Genuchten model for air/water capillary pressure based on an encore function for saturation

4.1.115 Capillary Pressure: Udell Cubic

Scope: Aria Material

Capillary Pressure: Udell Cubic [{of|species|subindex} *Species*]= Udell_Cubic [Residual_Wetting_P
= *residual_wetting_phase_saturation* |Residual_Nonwetting_Phase_Saturation = *residual_nonwetting_phase_s*
|Sigma = *sigma* |C1 = *c1* |C2 = *c2* |C3 = *c3* |Wetting_Phase = *wetting_phase*]

Parameter	Value	Default
Species	string	undefined
residual_wetting_phase_saturation	real	undefined
residual_nonwetting_phase_saturation	real	undefined
sigma	real	undefined
c1	real	undefined
c2	real	undefined
c3	real	undefined
wetting_phase	"string"	undefined

Summary A Udell cubic model for air/water capillary pressure

4.1.116 Capillary Pressure: Computed

Scope: Aria Material

Capillary Pressure: Computed [{of|species|subindex} *Species*]= Computed [Wetting_Phase
= *wetting_phase* |Nonwetting_Phase = *nonwetting_phase*]

Parameter	Value	Default
Species	string	undefined
wetting_phase	"string"	undefined
nonwetting_phase	"string"	undefined

Summary Capillary pressure computed based on the pressure difference between the non-wetting and wetting phases, each of which may be specified as parameters.

4.1.117 Capillary Pressure: Brooks Corey

Scope: Aria Material

Capillary Pressure: Brooks Corey [{of|species|subindex} *Species*]= Brooks_Corey [Wetting_Phase
= *wetting_phase*]

Parameter	Value	Default
Species	string	undefined
wetting_phase	"string"	undefined

Summary Capillary pressure computed based on the Brooks-Corey model

4.1.118 Characteristic Length: Correlation

Scope: Aria Material

Characteristic Length: Correlation [{of | species | subindex} *Species*]= Correlation [Dh = *Dh* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
Dh	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation characteristic length with constant value

4.1.119 Characteristic Length: Correlation Bar Perimeter

Scope: Aria Material

Characteristic Length: Correlation Bar Perimeter [{of | species | subindex} *Species*]= Correlation_Ba
[Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
bulk_node_interface	"string"	undefined

Summary Correlation characteristic length with bar perimeter model

4.1.120 Characteristic Length: Correlation Hydraulic Diameter

Scope: Aria Material

Characteristic Length: Correlation Hydraulic Diameter [{of | species | subindex} *Species*]= Correlation_Hydraulic_Diameter [Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
bulk_node_interface	"string"	undefined

Summary Correlation characteristic length with hydraulic diameter model

4.1.121 Characteristic Length: Correlation Wetted Perimeter

Scope: Aria Material

Characteristic Length: Correlation Wetted Perimeter [{of | species | subindex} *Species*]= Correlation_Wetted_Perimeter [Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
bulk_node_interface	"string"	undefined

Summary Correlation characteristic length with wetted perimeter model

4.1.122 Charge Density

Scope: Aria Material

`Charge Density [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	`string`	`undefined`
value	`real`	`undefined`

Summary Constant value

4.1.123 Charge Density: Copied

Scope: Aria Material

`Charge Density: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.124 Charge Density: User Plugin

Scope: Aria Material

`Charge Density: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.125 Chemical Potential: Ideal Solution

Scope: Aria Material

`Chemical Potential: Ideal Solution [ {of | species | subindex} Species ]= Ideal_Solution [ R = r | Temperature_Material_Phase = temperature_material_phase ]`

Parameter	Value	Default
Species	`string`	`undefined`
r	`real`	`undefined`
temperature_material_phase	`"string"`	`undefined`

Summary Ideal solution chemical potential, $\mu = RT\log(C)$.

4.1.126 Chemical Potential: From Equilibrium Potential

Scope: Aria Material

`Chemical Potential: From Equilibrium Potential [ {of|species|subindex} Species ]= From_Equilibrium [ F = f ]`

Parameter	Value	Default
Species	string	undefined
f	real	undefined

Summary Chemical potential from the equilibrium voltage of an electrochemical reaction.This model assumes that it is a single electron intercalation reaction and that only differences in the chemical potential are relevant, not the absolute magnitude. Therefore $\mu = -F\phi_{eq}$.

4.1.127 Chemical Potential: Isotropic Mesh Stress

Scope: Aria Material

`Chemical Potential: Isotropic Mesh Stress [ {of|species|subindex} Species ]= Isotropic_Mesh_Stress [ Strain_Coeff = strain_coeff ]`

Parameter	Value	Default
Species	string	undefined
strain_coeff	real	undefined

Summary Contribution to the chemical potential from species strain. Assumes that the strain is isotropic so $\mu = \Omega\sigma_{kk}/3$ where Ω is the volumetric expansion coefficient and σ is the stress tensor. This model uses the mesh stress provided by a projection equation in order to recover stress gradients.

4.1.128 Contact Electrical Conductance Coefficient

Scope: Aria Material

`Contact Electrical Conductance Coefficient [ {of|species|subindex} Species ]= Constant [ H = h ]`

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.129 Contact Electrical Conductance Coefficient: Copied

Scope: Aria Material

`Contact Electrical Conductance Coefficient: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.130 Contact Electrical Conductance Coefficient: User Plugin

Scope: Aria Material

Contact Electrical Conductance Coefficient: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.131 Contact Electrical Conductance Coefficient: Calore User Sub

Scope: Aria Material

Contact Electrical Conductance Coefficient: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.132 Contact Heat Transfer Coefficient

Scope: Aria Material

Contact Heat Transfer Coefficient [{of|species|subindex} *Species*]= Constant [H = *h*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.133 Contact Heat Transfer Coefficient: Copied

Scope: Aria Material

Contact Heat Transfer Coefficient: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.134 Contact Heat Transfer Coefficient: User Plugin

Scope: Aria Material

Contact Heat Transfer Coefficient: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.135 Contact Heat Transfer Coefficient: Calore User Sub

Scope: Aria Material

Contact Heat Transfer Coefficient: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Name = *name* | Type = *type* | Multiplier = *multiplier* | Material_Data_Block = *material_data_block* | Data = *data* | Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.136 Continuity Diffusion Coefficient

Scope: Aria Material

Continuity Diffusion Coefficient [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.137 Continuity Diffusion Coefficient: Copied

Scope: Aria Material

`Continuity Diffusion Coefficient: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.138 Continuity Diffusion Coefficient: User Plugin

Scope: Aria Material

`Continuity Diffusion Coefficient: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.139 Continuity Diffusion Coefficient: Shakib

Scope: Aria Material

`Continuity Diffusion Coefficient: Shakib [ {of | species | subindex} Species ]= Shakib [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Pressure POISSON with diffusion coefficient given by Shakib tau.

4.1.140 Continuity Diffusive Flux: Basic

Scope: Aria Material

`Continuity Diffusive Flux: Basic [ {of | species | subindex} Species ]= Basic [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Basic model of continuity diffusive flux

4.1.141 Continuity Diffusive Flux: Pressure Poisson

Scope: Aria Material

Continuity Diffusive Flux: Pressure Poisson [{of | species | subindex} *Species*]= Pressure_Poisson [ ]

Parameter	Value	Default
Species	string	undefined

Summary Pressure poisson model of continuity diffusive flux

4.1.142 Continuity Face Stabilization Scaling: Default

Scope: Aria Material

Summary Default face stabilization scaling for continuity equation

4.1.143 Cte

Scope: Aria Material

Cte [{of | species | subindex} *Species*]= Constant [Cte = *cte*]

Parameter	Value	Default
Species	string	undefined
cte	real	undefined

Summary Constant value

4.1.144 Cte: Copied

Scope: Aria Material

Cte: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.145 Cte: User Plugin

Scope: Aria Material

Cte: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_str*
]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.146 Current Density

Scope: Aria Material

Current Density [{of|species|subindex} *Species*]= Constant [Valuex = *valuex* |Valuey
= *valuey* |Valuez = *valuez*]

Parameter	Value	Default
Species	string	undefined
valuex	real	undefined
valuey	real	undefined
valuez	real	undefined

Summary Constant value

4.1.147 Current Density: User Plugin

Scope: Aria Material

Current Density: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name*
|*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.148 Current Density: Ohms Law

Scope: Aria Material

Current Density: Ohms Law [{of|species|subindex} *Species*]= Ohms_Law [ ]

Parameter	Value	Default
Species	string	undefined

Summary Ohms law current density

4.1.149 Current Density: Basic

Scope: Aria Material

`Current Density: Basic [ {of | species | subindex} Species ]= Basic [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Basic current density

4.1.150 Current Density: Electrolyte

Scope: Aria Material

`Current Density: Electrolyte [ {of | species | subindex} Species ]= Electrolyte [ F = f ]`

Parameter	Value	Default
Species	string	undefined
f	real	undefined

Summary Electrolyte current density

4.1.151 Current Density: Dielectric Displacement

Scope: Aria Material

`Current Density: Dielectric Displacement [ {of | species | subindex} Species ]= Dielectric_Displacemen`
`[ ]`

Parameter	Value	Default
Species	string	undefined

Summary Dielectric displacement current density

4.1.152 Current Density: Thermoelectric

Scope: Aria Material

`Current Density: Thermoelectric [ {of | species | subindex} Species ]= Thermoelectric [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Thermoelectric current density

4.1.153 Current Exchange Density

Scope: Aria Material

`Current Exchange Density [ {of | species | subindex} Species ]= Constant [ I_0 = i_0 ]`

Parameter	Value	Default
Species	string	undefined
i_0	real	undefined

Summary Constant value

4.1.154 Current Exchange Density: Copied

Scope: Aria Material

`Current Exchange Density: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.155 Current Exchange Density: User Plugin

Scope: Aria Material

`Current Exchange Density: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.156 Cantera Xml File

Scope: Aria Material

`Cantera Xml File {=|are|is} XML Path`

Parameter	Value	Default
XML Path	string	undefined

Description This specifies the name of the XML file used by Cantera to import the properties of the chemical species mixture. The order of the species in the simulation will be set by the order in this file.

4.1.157 Decay Constant: Hdiff

Scope: Aria Material

`Decay Constant: Hdiff [ {of|species|subindex} Species ]= Hdiff [ Decay = decay ]`

Parameter	Value	Default
Species	string	undefined
decay	real	undefined

Summary Decay constant used for hydrogen transport

4.1.158 Density

Scope: Aria Material

```
Density [ {of|species|subindex} Species ]= Constant [ Rho = rho ]
```

Parameter	Value	Default
Species	string	undefined
rho	real	undefined

Summary Constant value

4.1.159 Density: Copied

Scope: Aria Material

```
Density: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.160 Density: User Plugin

Scope: Aria Material

```
Density: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_
]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parts	"string"	undefined

Summary Value from a user plugin

4.1.161 Density: Calore User Sub

Scope: Aria Material

```
Density: Calore User Sub [ {of|species|subindex} Species ]= Calore_User_Sub [ Name = name
|Type = type |Multiplier = multiplier |Material_Data_Block = material_data_block |Data =
data |Scaling_Field = scaling_field ]
```

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.162 Density: Cantera

Scope: Aria Material

Density: Cantera [{of | species | subindex} *Species*]= Cantera [Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
poffset	real	undefined

Summary Cantera density summary

4.1.163 Density: Cantera Molten Salt

Scope: Aria Material

Density: Cantera Molten Salt [{of | species | subindex} *Species*]= Cantera_Molten_Salt [Densityconversion = *densityConversion* | Canteraxmlfile = *canteraXMLFile*]

Parameter	Value	Default
Species	string	undefined
densityConversion	real	undefined
canteraXMLFile	"string"	undefined

Summary Cantera density summary

4.1.164 Density: Clsm

Scope: Aria Material

Density: Clsm [{of | species | subindex} *Species*]= Clsm [Primaryproperty = *primaryProperty* | Secondaryproperty = *secondaryProperty*]

Parameter	Value	Default
Species	string	undefined
primaryProperty	real	undefined
secondaryProperty	real	undefined

Summary CLSM density

4.1.165 Density: Compressible Boussinesq

Scope: Aria Material

Density: Compressible Boussinesq [{of|species|subindex} *Species*]= Compressible_Boussinesq [Ref_Density = *ref_density* |Alpha = *alpha* |Ref_Pressure = *ref_pressure* |Ref_Temperature = *ref_temperature*]

Parameter	Value	Default
Species	string	undefined
ref_density	real	undefined
alpha	real	undefined
ref_pressure	real	undefined
ref_temperature	real	undefined

Summary A compressible Boussinesq model for the density

4.1.166 Density: Curing Foam

Scope: Aria Material

Density: Curing Foam [{of|species|subindex} *Species*]= Curing_Foam [Vfrac_Subindex = *vfrac_subindex* |Rho_E = *rho_e* |Rho_F = *rho_f* |R = *r* |Phi_Zero = *phi_zero*]

Parameter	Value	Default
Species	string	undefined
vfrac_subindex	integer	undefined
rho_e	real	undefined
rho_f	real	undefined
r	real	undefined
phi_zero	real	undefined

4.1.167 Density: Exp Decay

Scope: Aria Material

Density: Exp Decay [{of|species|subindex} *Species*]= Exp_Decay [Rho_Initial = *rho_initial* |Rho_Final = *rho_final* |K = *k*]

Parameter	Value	Default
Species	string	undefined
rho_initial	real	undefined
rho_final	real	undefined
k	real	undefined

Summary Density as an exponential decay function in time

4.1.168 Density: Foam Time Temp

Scope: Aria Material

Density: Foam Time Temp [{of|species|subindex} *Species*]= Foam_Time_Temp [Rho_Initial = *rho_initial* |Rho_Final = *rho_final* |C_Param = *c_param* |D_Param = *d_param* |T_Max = *t_max* |Time_Naught = *time_naught*]

Parameter	Value	Default
Species	string	undefined
rho_initial	real	undefined
rho_final	real	undefined
c_param	real	undefined
d_param	real	undefined
t_max	real	undefined
time_naught	real	undefined

Summary Foam time temp density

4.1.169 Density: From Volume Fraction Gas

Scope: Aria Material

Density: From Volume Fraction Gas [{of|species|subindex} *Species*]= From_Volume_Fraction_Gas [ ]

Parameter	Value	Default
Species	string	undefined

Summary Foam density calculated from a weighting with the volume fraction of gas.

4.1.170 Density: From Bubble State

Scope: Aria Material

Density: From Bubble State [{of|species|subindex} *Species*]= From_Bubble_State [Gas_Species_Name = *gas_species_name*]

Parameter	Value	Default
Species	string	undefined
gas_species_name	"string"	undefined

Summary Density from foam bubble model

4.1.171 Density: General Ideal Gas

Scope: Aria Material

Density: General Ideal Gas [{of|species|subindex} *Species*]= General_Ideal_Gas [R = *r* |Pref = *pref* |Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
pref	real	undefined
poffset	real	undefined

Summary General ideal gas density

4.1.172 Density: General Ideal Gas Extract Average Pressure

Scope: Aria Material

Density: General Ideal Gas Extract Average Pressure [{of|species|subindex} *Species*]= General_Ideal_Gas_Extract_Average_Pressure [R = *r* |Pref = *pref* |Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
pref	real	undefined
poffset	real	undefined

Summary General ideal gas density that subtracts off the average pressure value as a constant every time step so that the pressure dof stays O(1) throughout the simulation.

4.1.173 Density: General Ideal Gas Thermodynamic Pressure

Scope: Aria Material

Density: General Ideal Gas Thermodynamic Pressure [{of|species|subindex} *Species*]= General_Ideal [R = *r* |Initial_Thermodynamic_Pressure = *initial_thermodynamic_pressure*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
initial_thermodynamic_pressure	real	undefined

Summary General ideal gas density that uses the thermodynamic pressure in the pressure-density relation, as in the low Mach equations.

4.1.174 Density: Ideal Gas

Scope: Aria Material

Density: Ideal Gas [{of|species|subindex} *Species*]= Ideal_Gas [ ]

Parameter	Value	Default
Species	string	undefined

Summary Ideal gas density

4.1.175 Density: Incompressible Ideal Gas

Scope: Aria Material

Density: Incompressible Ideal Gas [{of|species|subindex} *Species*]= Incompressible_Ideal_Gas [P_Ref = *p_ref* |T_Ref = *t_ref* |R = *r*]

Parameter	Value	Default
Species	string	undefined
p_ref	real	undefined
t_ref	real	undefined
r	real	undefined

Summary Incompressible ideal gas density

4.1.176 Density: Mixture Fraction

Scope: Aria Material

Density: Mixture Fraction [{of | species | subindex} *Species*]= Mixture_Fraction [Primaryproperty = *primaryProperty* |Secondaryproperty = *secondaryProperty*]

Parameter	Value	Default
Species	string	undefined
primaryProperty	real	undefined
secondaryProperty	real	undefined

4.1.177 Density: Phase Average

Scope: Aria Material

Density: Phase Average [{of | species | subindex} *Species*]= Phase_Average [Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.178 Density: Interpolated Phase Average

Scope: Aria Material

Density: Interpolated Phase Average [{of | species | subindex} *Species*]= Interpolated_Phase_Average [Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Interpolated Phase-averaged values

4.1.179 Density: Single Component Ideal Gas

Scope: Aria Material

Density: Single Component Ideal Gas [{of|species|subindex} *Species*]= Single_Component_Ideal_Gas [P_Ref = *p_ref* |T_Ref = *t_ref* |M = *m* |R = *r*]

Parameter	Value	Default
Species	string	undefined
p_ref	real	undefined
t_ref	real	undefined
m	real	undefined
r	real	undefined

4.1.180 Density: Single Component Adiabatic Ideal Gas

Scope: Aria Material

Density: Single Component Adiabatic Ideal Gas [{of|species|subindex} *Species*]= Single_Component_A [P_Ref = *p_ref* |P_0 = *p_0* |Density_0 = *density_0* |Gamma = *gamma*]

Parameter	Value	Default
Species	string	undefined
p_ref	real	undefined
p_0	real	undefined
density_0	real	undefined
gamma	real	undefined

4.1.181 Density: Stanford

Scope: Aria Material

Density: Stanford [{of|species|subindex} *Species*]= Stanford [N = *n*]

Parameter	Value	Default
Species	string	undefined
n	real	undefined

4.1.182 Density: Thermal

Scope: Aria Material

Density: Thermal [{of|species|subindex} *Species*]= Thermal [A = *a* |B = *b* |C = *c* |D = *d*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined
b	real	undefined
c	real	undefined
d	real	undefined

Summary Density as a cubic polynomial in temperature

4.1.183 Density: Mass Average

Scope: Aria Material

Density: Mass Average [{of | species | subindex} *Species*]= Mass_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Density as a mass average of the species bulk densities ($\rho = \sum_i \left(\frac{Y_i}{\rho_{b,i}}\right)^{-1}$.

4.1.184 Density: Mass Preserving

Scope: Aria Material

Density: Mass Preserving [{of | species | subindex} *Species*]= Mass_Preserving [Rho = *rho*]

Parameter	Value	Default
Species	string	undefined
rho	real	undefined

Summary Density applied as a ratio of original to current element volume.

4.1.185 Density: Concentration Average

Scope: Aria Material

Density: Concentration Average [{of | species | subindex} *Species*]= Concentration_Average [Fluid_Density = *fluid_density* |Lambda = *lambda*]

Parameter	Value	Default
Species	string	undefined
fluid_ density	real	undefined
lambda	real	undefined

Summary Density as a concentration (phi) average

4.1.186 Density: From Mass Fraction

Scope: Aria Material

Density: From Mass Fraction [{of | species | subindex} *Species*]= From_Mass_Fraction [ ]

Parameter	Value	Default
Species	string	undefined

Summary Species density computed from mass fraction

4.1.187 Density: Adaptive Table Lookup

Scope: Aria Material

Density: Adaptive Table Lookup [{of | species | subindex} *Species*]= Adaptive_Table_Lookup [File_Name = *file_name*]

Parameter	Value	Default
Species	string	undefined
file_name	"string"	undefined

Summary An adaptive table lookup of density

4.1.188 Density: Porous Phase Average

Scope: Aria Material

Density: Porous Phase Average [{of | species | subindex} *Species*]= Porous_Phase_Average [Liquid_Phase_Name = *Liquid_Phase_Name* | Gas_Phase_Name = *Gas_Phase_Name*]

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined

Summary Density for a multiphase material of two or three phases

4.1.189 Density: From Chemeq

Scope: Aria Material

Density: From Chemeq [{of | species | subindex} *Species*]= From_Chemeq [ ]

Parameter	Value	Default
Species	string	undefined

Summary Density of one species extracted from ChemEq

4.1.190 Density: From Chemeq Solids

Scope: Aria Material

Density: From Chemeq Solids [{of | species | subindex} *Species*]= From_Chemeq_Solids [Rhob0 = *rhob0*]

Parameter	Value	Default
Species	string	undefined
rhob0	real	undefined

Summary Condensed phase density from ChemEq species concentrations for solid species

4.1.191 Density: Sum All Species

Scope: Aria Material

Density: Sum All Species [{of|species|subindex} *Species*]= Sum_All_Species [ ]

Parameter	Value	Default
Species	string	undefined

4.1.192 Density: Fortran

Scope: Aria Material

Density: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine density.

4.1.193 Dispersed Phase Density

Scope: Aria Material

Dispersed Phase Density [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.194 Dispersed Phase Density: Copied

Scope: Aria Material

Dispersed Phase Density: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.195 Dispersed Phase Density: User Plugin

Scope: Aria Material

Dispersed Phase Density: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.196 Dispersed Phase Momentum Stress: Newtonian

Scope: Aria Material

Dispersed Phase Momentum Stress: Newtonian [{of | species | subindex} *Species*]= Newtonian [Cts = *cts*]

Parameter	Value	Default
Species	string	undefined
cts	real	undefined

4.1.197 Dispersed Phase Velocity

Scope: Aria Material

Dispersed Phase Velocity [{of | species | subindex} *Species*]= Constant [Velocity_X = *Velocity_X* | Velocity_Y = *Velocity_Y* | Velocity_Z = *Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
Velocity_X	real	undefined
Velocity_Y	real	undefined
Velocity_Z	real	undefined

Summary Constant value

4.1.198 Dispersed Phase Velocity: User Plugin

Scope: Aria Material

Dispersed Phase Velocity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.199 Dispersed Phase Volume Fraction

Scope: Aria Material

`Dispersed Phase Volume Fraction [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.200 Dispersed Phase Volume Fraction: Copied

Scope: Aria Material

`Dispersed Phase Volume Fraction: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.201 Dispersed Phase Volume Fraction: User Plugin

Scope: Aria Material

`Dispersed Phase Volume Fraction: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.202 Distribution Coefficient

Scope: Aria Material

`Distribution Coefficient [ {of | species | subindex} Species ]= Constant [ Kd = kd ]`

Parameter	Value	Default
Species	string	undefined
kd	real	undefined

Summary Constant value

4.1.203 Distribution Coefficient: Copied

Scope: Aria Material

Distribution Coefficient: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.204 Distribution Coefficient: User Plugin

Scope: Aria Material

Distribution Coefficient: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.205 Distribution Coefficient: Langmuir

Scope: Aria Material

Distribution Coefficient: Langmuir [{of | species | subindex} *Species*]= Langmuir [K = *k* | K_I = *k_I*]

Parameter	Value	Default
Species	string	undefined
k	real	undefined
k_I	real	undefined

Summary Langmuir isotherm distribution coefficient for species equation

4.1.206 Distribution Coefficient: Competitive Langmuir

Scope: Aria Material

Distribution Coefficient: Competitive Langmuir [{of | species | subindex} *Species*]= Competitive_Langm [K_I = *k_i* | K_J = *k_j* | K_Ie = *k_ie* | Species_J = *species_j*]

Parameter	Value	Default
Species	string	undefined
k_i	real	undefined
k_j	real	undefined
k_ie	real	undefined
species_j	integer	undefined

Summary Competitive Langmuir isotherm distribution coefficient for species equation

4.1.207 Effective Diffusivity: Hdiff

Scope: Aria Material

Effective Diffusivity: Hdiff [{of | species | subindex} *Species*]= Hdiff [K_T = *k_t* | N_L = *n_l*]

Parameter	Value	Default
Species	string	undefined
k_t	real	undefined
n_l	real	undefined

Summary ?

4.1.208 Electric Displacement: Linear

Scope: Aria Material

Electric Displacement: Linear [{of | species | subindex} *Species*]= Linear [ ]

Parameter	Value	Default
Species	string	undefined

Summary Linear electric displacement

4.1.209 Electric Displacement: Basic

Scope: Aria Material

Electric Displacement: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Basic electric displacement

4.1.210 Electric Displacement: Generalized

Scope: Aria Material

Electric Displacement: Generalized [{of | species | subindex} *Species*]= Generalized [ ]

Parameter	Value	Default
Species	string	undefined

Summary Generalized electric displacement

4.1.211 Electrical Conductivity

Scope: Aria Material

```
Electrical Conductivity [ {of|species|subindex} Species ]= Constant [ Sigma = sigma ]
```

Parameter	Value	Default
Species	string	undefined
sigma	real	undefined

Summary Constant value

4.1.212 Electrical Conductivity: Copied

Scope: Aria Material

```
Electrical Conductivity: Copied [ {of|species|subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.213 Electrical Conductivity: User Plugin

Scope: Aria Material

```
Electrical Conductivity: User Plugin [ {of|species|subindex} Species ]= User_Plugin [
Name = Name |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.214 Electrical Conductivity: From Resistivity

Scope: Aria Material

```
Electrical Conductivity: From Resistivity [ {of|species|subindex} Species ]= From_Resistivity
[ ]
```

Parameter	Value	Default
Species	string	undefined

Summary Electrical conductivity calculated from the electrical resistance

4.1.215 Electrical Conductivity: Tbc

Scope: Aria Material

Electrical Conductivity: Tbc [{of | species | subindex} *Species*]= Tbc [Ki = *ki* | Ti = *ti* | E = *e* | R = *r*]

Parameter	Value	Default
Species	string	undefined
ki	real	undefined
ti	real	undefined
e	real	undefined
r	real	undefined

Summary TBC model for electrical conductivity

4.1.216 Electrical Conductivity: Thermal

Scope: Aria Material

Electrical Conductivity: Thermal [{of | species | subindex} *Species*]= Thermal [A = *a* | B = *b* | C = *c* | D = *d*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined
b	real	undefined
c	real	undefined
d	real	undefined

Summary Electrical conductivity as a cubic polynomial in temperature

4.1.217 Electrical Conductivity: Bruggeman Volume Averaged

Scope: Aria Material

Electrical Conductivity: Bruggeman Volume Averaged [{of | species | subindex} *Species*]= Bruggeman_Volume_Averaged [Exponent = *exponent*]

Parameter	Value	Default
Species	string	undefined
exponent	real	undefined

Summary This model averages the properties of each species in the same material phase based on their volume fraction assuming a Bruggeman tortuosity model.

4.1.218 Electrical Conductivity: Arrhenius

Scope: Aria Material

`Electrical Conductivity: Arrhenius [ {of|species|subindex} Species ]= Arrhenius [ R = r |Sigma0 = sigma0 |A = a |Ea = Ea ]`

Parameter	Value	Default
Species	string	undefined
r	real	undefined
sigma0	real	undefined
a	real	undefined
Ea	real	undefined

Summary Arrhenius electrical conductivity, $\sigma = \sigma_0 + A * exp\left(-\frac{E_a}{/}RT\right)$

4.1.219 Electrical Conductivity: Electrode

Scope: Aria Material

`Electrical Conductivity: Electrode [{of|species|subindex} Species]= Electrode [Electrodefile = electrodeFile |F = f |Kmolconversion = kmolConversion |Jconversion = JConversion |Meterconversion = meterConversion |Mincapacity = minCapacity |Deactivationvoltage = deactivationVoltage |Voltagevariablename = voltageVariableName |Abortaction = abortAction |Inactivesolidfraction = inactiveSolidFraction |Pref = Pref |Sens_Eps = Sens_eps |Sens_Order = Sens_order |A = a |B = b |C = c |Min = min |Max = max]`

Parameter	Value	Default
Species	string	undefined
electrodeFile	"string"	undefined
f	real	undefined
kmolConversion	real	undefined
JConversion	real	undefined
meterConversion	real	undefined
minCapacity	real	undefined
deactivationVoltage	real	undefined
voltageVariableName	"string"	undefined
abortAction	"string"	undefined
inactiveSolidFraction	real	undefined
Pref	real	undefined
Sens_eps	real	undefined
Sens_order	"string"	undefined
a	real	undefined
b	real	undefined
c	real	undefined
min	real	undefined
max	real	undefined

Summary Electrical conductivity as a polynomial function of electrode capacity remaining with $\sigma = A + BF + CF^2$ where F is the remaining electrode capacity in Amp-s. A min and max value must also be provided to bound the conductivity value.

4.1.220 Electrical Permittivity

Scope: Aria Material

`Electrical Permittivity [ {of|species|subindex} Species ]= Constant [ Kappa = kappa ]`

Parameter	Value	Default
Species	string	undefined
kappa	real	undefined

Summary Constant value

4.1.221 Electrical Permittivity: Copied

Scope: Aria Material

`Electrical Permittivity: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.222 Electrical Permittivity: User Plugin

Scope: Aria Material

`Electrical Permittivity: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.223 Electrical Resistance

Scope: Aria Material

`Electrical Resistance [ {of|species|subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.224 Electrical Resistance: Copied

Scope: Aria Material

`Electrical Resistance: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.225 Electrical Resistance: User Plugin

Scope: Aria Material

`Electrical Resistance: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.226 Electrical Resistivity

Scope: Aria Material

`Electrical Resistivity [ {of|species|subindex} Species ]= Constant [ Rho = rho ]`

Parameter	Value	Default
Species	string	undefined
rho	real	undefined

Summary Constant value

4.1.227 Electrical Resistivity: Copied

Scope: Aria Material

`Electrical Resistivity: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.228 Electrical Resistivity: User Plugin

Scope: Aria Material

Electrical Resistivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.229 Electrical Resistivity: From Conductivity

Scope: Aria Material

Electrical Resistivity: From Conductivity [{of | species | subindex} *Species*]= From_Conductivity [ ]

Parameter	Value	Default
Species	string	undefined

Summary Electrical resistivity calculated from the electrical conductivity

4.1.230 Electromigration Coefficient

Scope: Aria Material

Electromigration Coefficient [{of | species | subindex} *Species*]= Constant [M = *m*]

Parameter	Value	Default
Species	string	undefined
m	real	undefined

Summary Constant value

4.1.231 Electromigration Coefficient: Copied

Scope: Aria Material

Electromigration Coefficient: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.232 Electromigration Coefficient: User Plugin

Scope: Aria Material

`Electromigration Coefficient: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.233 Emissivity

Scope: Aria Material

`Emissivity [ {of|species|subindex} Species ]= Constant [ E = e ]`

Parameter	Value	Default
Species	`string`	`undefined`
e	`real`	`undefined`

Summary Constant value

4.1.234 Emissivity: Copied

Scope: Aria Material

`Emissivity: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.235 Emissivity: User Plugin

Scope: Aria Material

`Emissivity: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.236 Emissivity: Calore User Sub

Scope: Aria Material

Emissivity: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.237 Emissivity: Fortran

Scope: Aria Material

Emissivity: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine emissivity.

4.1.238 Emissivity: Banded Wavelength

Scope: Aria Material

Emissivity: Banded Wavelength [{of|species|subindex} *Species*]= Banded_Wavelength [E0 = *e0* |E1 = *e1* |E2 = *e2* |E3 = *e3* |E4 = *e4* |E5 = *e5* |E6 = *e6* |E7 = *e7* |Wavelength_Band_Model = *wavelength_band_model*]

Parameter	Value	Default
Species	string	undefined
e0	real	undefined
e1	real	undefined
e2	real	undefined
e3	real	undefined
e4	real	undefined
e5	real	undefined
e6	real	undefined
e7	real	undefined
wavelength_band_model	"string"	undefined

Summary Banded wavelength emissivity.

4.1.239 Enclosure Mbk

Scope: Aria Material

`Enclosure Mbk [ {of | species | subindex} Species ]= Constant [ K = k ]`

Parameter	Value	Default
Species	string	undefined
k	real	undefined

Summary Constant value

4.1.240 Enclosure Mbk: Copied

Scope: Aria Material

`Enclosure Mbk: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.241 Enclosure Mbk: User Plugin

Scope: Aria Material

`Enclosure Mbk: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.242 Enclosure Mbk: Calore User Sub

Scope: Aria Material

Enclosure Mbk: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.243 Enclosure Mbl

Scope: Aria Material

Enclosure Mbl [{of|species|subindex} *Species*]= Constant [L = *l*]

Parameter	Value	Default
Species	string	undefined
l	real	undefined

Summary Constant value

4.1.244 Enclosure Mbl: Copied

Scope: Aria Material

Enclosure Mbl: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.245 Enclosure Mbl: User Plugin

Scope: Aria Material

Enclosure Mbl: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parameter	"string"	undefined

Summary Value from a user plugin

4.1.246 Enclosure Mbl: Calore User Sub

Scope: Aria Material

Enclosure Mbl: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.247 Enclosure Mbl: From Geometry

Scope: Aria Material

Enclosure Mbl: From Geometry [{of|species|subindex} *Species*]= From_Geometry [Prefactor = *prefactor* |Enclosure = *enclosure*]

Parameter	Value	Default
Species	string	undefined
prefactor	real	undefined
enclosure	"string"	undefined

Summary Values from geometry with a user selectable pre-factor

4.1.248 Energy Diffusive Flux: Basic

Scope: Aria Material

Energy Diffusive Flux: Basic [{of|species|subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Basic model of energy diffusive flux

4.1.249 Energy Diffusive Flux: From Mixture Mass Diffusivity

Scope: Aria Material

`Energy Diffusive Flux: From Mixture Mass Diffusivity [ {of|species|subindex} Species ]= From_Mixture_Mass_Diffusivity [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Energy diffusive flux = -density * mixture_mass_diffusivity * grad(enthalpy)

4.1.250 Energy Diffusive Flux: Energy Mass

Scope: Aria Material

`Energy Diffusive Flux: Energy Mass [ {of|species|subindex} Species ]= Energy_Mass [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Energy mass model of energy diffusive flux

4.1.251 Energy Face Stabilization Scaling: Default

Scope: Aria Material

Summary Default face stabilization scaling for energy equation

4.1.252 Enthalpy

Scope: Aria Material

`Enthalpy [ {of|species|subindex} Species ]= Constant [ H = h ]`

Parameter	Value	Default
Species	`string`	`undefined`
h	`real`	`undefined`

Summary Constant value

4.1.253 Enthalpy: Copied

Scope: Aria Material

`Enthalpy: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.254 Enthalpy: User Plugin

Scope: Aria Material

Enthalpy: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.255 Enthalpy: Cantera

Scope: Aria Material

Enthalpy: Cantera [{of | species | subindex} *Species*]= Cantera [Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
poffset	real	undefined

Summary Cantera enthalpy

4.1.256 Enthalpy: From Temperature

Scope: Aria Material

Enthalpy: From Temperature [{of | species | subindex} *Species*]= From_Temperature [T = *t* | Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
t	real	undefined
poffset	real	undefined

Summary Cantera enthalpy computed from temperature

4.1.257 Enthalpy: Mass Average

Scope: Aria Material

Enthalpy: Mass Average [{of | species | subindex} *Species*]= Mass_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Enthalpy computed from mass average of species enthalpies

4.1.258 Enthalpy Advection: Porous

Scope: Aria Material

`Enthalpy Advection: Porous [ {of|species|subindex} Species ]= Porous [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Advected quantity for porous enthalpy equation

4.1.259 Enthalpy Advection: Porous Upwind

Scope: Aria Material

`Enthalpy Advection: Porous Upwind [ {of|species|subindex} Species ]= Porous_Upwind [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Upwinded advected quantity for porous enthalpy equation. Only valid for CVFEM.

4.1.260 Entrance Distance: Correlation Bar

Scope: Aria Material

`Entrance Distance: Correlation Bar [ {of|species|subindex} Species ]= Correlation_Bar [ Phase = phase |Gas_Exponent = gas_exponent |Bulk_Node_Interface = bulk_node_interface ]`

Parameter	Value	Default
Species	string	undefined
phase	"string"	undefined
gas_exponent	real	undefined
bulk_node_interface	"string"	undefined

Summary Correction factor for difference between near-wall and bulk fluid state for Gnielinski pipe and annulus correlations.

4.1.261 Entrance Length: Correlation

Scope: Aria Material

`Entrance Length: Correlation [ {of|species|subindex} Species ]= Correlation [ De = De |Bulk_Node_Interface = bulk_node_interface ]`

Parameter	Value	Default
Species	string	undefined
De	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Entrance Length

4.1.262 Entry Pressure

Scope: Aria Material

`Entry Pressure [ {of|species|subindex} `*`Species`*` ]= Constant [ Pe = `*`pe`*` ]`

Parameter	Value	Default
Species	string	undefined
pe	real	undefined

Summary Constant value

4.1.263 Entry Pressure: Copied

Scope: Aria Material

`Entry Pressure: Copied [ {of|species|subindex} `*`Species`*` ]= Copied [ Source = `*`source`*` ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.264 Entry Pressure: User Plugin

Scope: Aria Material

`Entry Pressure: User Plugin [ {of|species|subindex} `*`Species`*` ]= User_Plugin [ Name = `*`Name`*` |`*`magic_trailing_string_vector_param`*` ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.265 Equilibrium Constant: Arrhenius

Scope: Aria Material

`Equilibrium Constant: Arrhenius [ {of|species|subindex} `*`Species`*` ]= Arrhenius [ C = `*`c`*` | M = `*`m`*` |W_B = `*`w_b`*` |R = `*`r`*` ]`

Parameter	Value	Default
Species	string	undefined
c	real	undefined
m	real	undefined
w_b	real	undefined
r	real	undefined

Summary ?

4.1.266 Equilibrium Potential

Scope: Aria Material

`Equilibrium Potential [ {of|species|subindex} Species ]= Constant [ U = u ]`

Parameter	Value	Default
Species	string	undefined
u	real	undefined

Summary Constant value

4.1.267 Equilibrium Potential: Copied

Scope: Aria Material

`Equilibrium Potential: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.268 Equilibrium Potential: User Plugin

Scope: Aria Material

`Equilibrium Potential: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.269 Equilibrium Potential: Nernst

Scope: Aria Material

Equilibrium Potential: Nernst [{of|species|subindex} *Species*]= Nernst [Species_A = *species_a* |Species_B = *species_b* |R = *r* |F = *f* |U = *u*]

Parameter	Value	Default
Species	string	undefined
species_a	"string"	undefined
species_b	"string"	undefined
r	real	undefined
f	real	undefined
u	real	undefined

Summary Nernst-Einstein form of the equilibrium potential

4.1.270 Equilibrium Potential: From Chemical Potential

Scope: Aria Material

Equilibrium Potential: From Chemical Potential [{of|species|subindex} *Species*]= From_Chemical_Po
[F = *f* |Species_Name = *species_name* |Species_Phase = *species_phase*]

Parameter	Value	Default
Species	string	undefined
f	real	undefined
species_name	"string"	undefined
species_phase	"string"	undefined

Summary Determine the equilbrium voltage based on the chemical potential of a selected species where $\phi_{equil} = -\mu/F$.

4.1.271 Equilibrium Potential: Two Phase

Scope: Aria Material

Equilibrium Potential: Two Phase [{of|species|subindex} *Species*]= Two_Phase [Cmin = *Cmin* |Cmax = *Cmax* |R = *r* |Species_Name = *species_name*]

Parameter	Value	Default
Species	string	undefined
Cmin	real	undefined
Cmax	real	undefined
r	real	undefined
species_name	"string"	undefined

Summary Equilibrium potential that is constant between Cmin and Cmax corresponding to a two-phase region. Outside of that range it behaves as an ideal solution.

4.1.272 Equilibrium Potential: Lcoo2

Scope: Aria Material

Equilibrium Potential: Lcoo2 [{of|species|subindex} *Species*]= Lcoo2 [ ]

Parameter	Value	Default
Species	string	undefined

Summary Equilibrium potential

4.1.273 Equilibrium Pressure

Scope: Aria Material

```
Equilibrium Pressure [ {of|species|subindex} Species ]= Constant [ P = p ]
```

Parameter	Value	Default
Species	string	undefined
p	real	undefined

Summary Constant value

4.1.274 Equilibrium Pressure: Copied

Scope: Aria Material

```
Equilibrium Pressure: Copied [ {of|species|subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.275 Equilibrium Pressure: User Plugin

Scope: Aria Material

```
Equilibrium Pressure: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name
= Name |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.276 Exchange Current Density

Scope: Aria Material

```
Exchange Current Density [ {of|species|subindex} Species ]= Constant [ Value = value ]
```

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.277 Exchange Current Density: Copied

Scope: Aria Material

Exchange Current Density: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.278 Exchange Current Density: User Plugin

Scope: Aria Material

Exchange Current Density: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.279 Exchange Current Density: Rate Constant Species

Scope: Aria Material

Exchange Current Density: Rate Constant Species [{of | species | subindex} *Species*]= Rate_Constant_Sp [F = *f* | K = *k* | Liquid_Exponent = *liquid_exponent* | Solid_Exponent = *solid_exponent* | Cs_Max = *Cs_max* | Liquid_Species = *liquid_species* | Solid_Species = *solid_species*]

Parameter	Value	Default
Species	string	undefined
f	real	undefined
k	real	undefined
liquid_exponent	real	undefined
solid_exponent	real	undefined
Cs_max	real	undefined
liquid_species	"string"	undefined
solid_species	"string"	undefined

Summary Computes an exchange current density of the form $i_0 = kF \prod \text{liquid\_species}^{\text{liquid\_exponent}} \times \prod \text{solid\_species}^{\text{solid\_exponent}}$ where the species whose concentrations are included are user-specified parameters.

4.1.280 Extension Speed

Scope: Aria Material

`Extension Speed [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	`string`	`undefined`
value	`real`	`undefined`

Summary Constant value

4.1.281 Extension Speed: Copied

Scope: Aria Material

`Extension Speed: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.282 Extension Speed: User Plugin

Scope: Aria Material

`Extension Speed: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.283 Extension Speed Diffusive Flux: Definition

Scope: Aria Material

`Extension Speed Diffusive Flux: Definition [ {of | species | subindex} Species ]= Definition [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary The definition of the extension speed diffusive flux

4.1.284 Extension Speed Multiplier

Scope: Aria Material

`Extension Speed Multiplier [ {of|species|subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.285 Extension Speed Multiplier: Copied

Scope: Aria Material

`Extension Speed Multiplier: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.286 Extension Speed Multiplier: User Plugin

Scope: Aria Material

`Extension Speed Multiplier: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.287 Fission Cross Section

Scope: Aria Material

`Fission Cross Section [ {of|species|subindex} Species ]= Constant [ Fiss = fiss ]`

Parameter	Value	Default
Species	string	undefined
fiss	real	undefined

Summary Constant value

4.1.288 Fission Cross Section: Copied

Scope: Aria Material

Fission Cross Section: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.289 Fission Cross Section: User Plugin

Scope: Aria Material

Fission Cross Section: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.290 Fission Cross Section: Linearized

Scope: Aria Material

Fission Cross Section: Linearized [{of | species | subindex} *Species*]= Linearized [Sigma_0 = *sigma_0* | D_Sigma_D_Rho = *d_sigma_d_rho* | D_Sigma_D_T = *d_sigma_d_t* | T_Sigma_0 = *t_sigma_0* | Rho_Sigma_0 = *rho_sigma_0*]

Parameter	Value	Default
Species	string	undefined
sigma_0	real	undefined
d_sigma_d_rho	real	undefined
d_sigma_d_t	real	undefined
t_sigma_0	real	undefined
rho_sigma_0	real	undefined

Summary Linearized fission cross section

4.1.291 Flow Area

Scope: Aria Material

Flow Area [{of|species|subindex} *Species*]= Constant [A = *a*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined

Summary Constant value

4.1.292 Flow Area: Copied

Scope: Aria Material

Flow Area: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.293 Flow Area: User Plugin

Scope: Aria Material

Flow Area: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.294 Flow Area: Spatial User Function

Scope: Aria Material

Flow Area: Spatial User Function [{of|species|subindex} *Species*]= Spatial_User_Function [Threshold = *threshold* |Multiplier = *multiplier* |Name = *name* |Bar = *bar* |X = *x*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined

Summary Spatially varying flow area based on a coordinate dependent user function

4.1.295 Flow Area: Correlation Spatial User Function

Scope: Aria Material

Flow Area: Correlation Spatial User Function [{of | species | subindex} *Species*]= Correlation_Spatial [Threshold = *threshold* |Multiplier = *multiplier* |Name = *name* |Bar = *bar* |X = *x* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined
bulk_node_interface	"string"	undefined

Summary Correlation spatially varying flow area based on a coordinate dependent user function

4.1.296 Flowing Liquid Viscosity

Scope: Aria Material

Flowing Liquid Viscosity [{of | species | subindex} *Species*]= Constant [Mu = *mu*]

Parameter	Value	Default
Species	string	undefined
mu	real	undefined

Summary Constant value

4.1.297 Flowing Liquid Viscosity: Copied

Scope: Aria Material

Flowing Liquid Viscosity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.298 Flowing Liquid Viscosity: User Plugin

Scope: Aria Material

Flowing Liquid Viscosity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parser	"string"	undefined

Summary Value from a user plugin

4.1.299 Fluid Beta: Correlation

Scope: Aria Material

Fluid Beta: Correlation [{of | species | subindex} *Species*]= Correlation [Beta = *beta* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
beta	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Expansion Coefficient

4.1.300 Fluid Density: Correlation

Scope: Aria Material

Fluid Density: Correlation [{of | species | subindex} *Species*]= Correlation [Rho = *rho* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
rho	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation fluid temperature

4.1.301 Fluid Temperature

Scope: Aria Material

Fluid Temperature [{of | species | subindex} *Species*]= Constant [T = *t*]

Parameter	Value	Default
Species	string	undefined
t	real	undefined

Summary Constant value

4.1.302 Fluid Temperature: Copied

Scope: Aria Material

Fluid Temperature: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.303 Fluid Temperature: User Plugin

Scope: Aria Material

Fluid Temperature: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.304 Fluid Velocity Magnitude: Correlation

Scope: Aria Material

Fluid Velocity Magnitude: Correlation [{of|species|subindex} *Species*]= Correlation [V = *v* |Prereq_Model = *prereq_model* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
v	real	undefined
prereq_model	"string"	undefined
bulk_node_interface	"string"	undefined

Summary Correlation fluid velocity

4.1.305 Flux Vector: Summed

Scope: Aria Material

Flux Vector: Summed [{of|species|subindex} *Species*]= Summed [Contributions = *contributions*]

Parameter	Value	Default
Species	string	undefined
contributions	"string"	undefined

Summary Flux vector as a sum of user-specified flux vectors

4.1.306 Forchheimer Drag Coeff

Scope: Aria Material

`Forchheimer Drag Coeff [ {of | species | subindex} Species ]= Constant [ C_F = c_f ]`

Parameter	Value	Default
Species	string	undefined
c_f	real	undefined

Summary Constant value

4.1.307 Forchheimer Drag Coeff: Copied

Scope: Aria Material

`Forchheimer Drag Coeff: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.308 Forchheimer Drag Coeff: User Plugin

Scope: Aria Material

`Forchheimer Drag Coeff: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.309 Freestream Density

Scope: Aria Material

`Freestream Density [ {of | species | subindex} Species ]= Constant [ Fsdens = fsdens ]`

Parameter	Value	Default
Species	string	undefined
fsdens	real	undefined

Summary Constant value

4.1.310 Freestream Density: Copied

Scope: Aria Material

`Freestream Density: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.311 Freestream Density: User Plugin

Scope: Aria Material

`Freestream Density: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.312 Freestream Gamma

Scope: Aria Material

`Freestream Gamma [ {of|species|subindex} Species ]= Constant [ G = g ]`

Parameter	Value	Default
Species	`string`	`undefined`
g	`real`	`undefined`

4.1.313 Freestream Pressure

Scope: Aria Material

`Freestream Pressure [ {of|species|subindex} Species ]= Constant [ Fspres = fspres ]`

Parameter	Value	Default
Species	`string`	`undefined`
fspres	`real`	`undefined`

Summary Constant value

4.1.314 Freestream Pressure: Copied

Scope: Aria Material

Freestream Pressure: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.315 Freestream Pressure: User Plugin

Scope: Aria Material

Freestream Pressure: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.316 Freestream Temperature

Scope: Aria Material

Freestream Temperature [{of | species | subindex} *Species*]= Constant [Fstemp = *fstemp*]

Parameter	Value	Default
Species	string	undefined
fstemp	real	undefined

Summary Constant value

4.1.317 Freestream Temperature: Copied

Scope: Aria Material

Freestream Temperature: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.318 Freestream Temperature: User Plugin

Scope: Aria Material

Freestream Temperature: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.319 Friction Factor

Scope: Aria Material

Friction Factor [{of | species | subindex} *Species*]= Constant [F = *f*]

Parameter	Value	Default
Species	string	undefined
f	real	undefined

Summary Constant value

4.1.320 Friction Factor: Copied

Scope: Aria Material

Friction Factor: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.321 Friction Factor: User Plugin

Scope: Aria Material

Friction Factor: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.322 Friction Factor: Smooth Tube

Scope: Aria Material

Friction Factor: Smooth Tube [{of | species | subindex} *Species*]= Smooth_Tube [ ]

Parameter	Value	Default
Species	string	undefined

Summary Friction factor computed from smooth tube model

4.1.323 Friction Factor: Smooth Annulus

Scope: Aria Material

Friction Factor: Smooth Annulus [{of | species | subindex} *Species*]= Smooth_Annulus [R = *r*]

Parameter	Value	Default
Species	string	undefined
r	"string"	undefined

Summary Friction factor computed from smooth annulus model

4.1.324 Friction Factor: Correlation

Scope: Aria Material

Friction Factor: Correlation [{of | species | subindex} *Species*]= Correlation [F = *f* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
f	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation friction factor with constant value

4.1.325 Friction Factor: Correlation Smooth Tube

Scope: Aria Material

Friction Factor: Correlation Smooth Tube [{of | species | subindex} *Species*]= Correlation_Smooth_Tube [Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
bulk_node_interface	"string"	undefined

Summary Correlation friction factor with smooth tube model

4.1.326 Friction Factor: Correlation Smooth Annulus

Scope: Aria Material

Friction Factor: Correlation Smooth Annulus [{of | species | subindex} *Species*]= Correlation_Smooth_A
[Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
bulk_node_interface	"string"	undefined

Summary Correlation friction factor with smooth annulus model

4.1.327 Gamma

Scope: Aria Material

Gamma [{of | species | subindex} *Species*]= Constant [G = *g*]

Parameter	Value	Default
Species	string	undefined
g	real	undefined

4.1.328 Gamma Dot

Scope: Aria Material

Gamma Dot [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.329 Gamma Dot: Copied

Scope: Aria Material

Gamma Dot: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.330 Gamma Dot: User Plugin

Scope: Aria Material

Gamma Dot: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* |
magic_trailing_string_vector_param]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.331 Gap Conductance Coefficient

Scope: Aria Material

Gap Conductance Coefficient [{of | species | subindex} *Species*]= Constant [H = *h*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.332 Gap Conductance Coefficient: Copied

Scope: Aria Material

Gap Conductance Coefficient: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.333 Gap Conductance Coefficient: User Plugin

Scope: Aria Material

Gap Conductance Coefficient: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.334 Gap Conductance Coefficient: Calore User Sub

Scope: Aria Material

Gap Conductance Coefficient: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Calore user subroutine contact gap conductance

4.1.335 Gap Conductance Coefficient: Fortran

Scope: Aria Material

Gap Conductance Coefficient: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran contact gap conductance subroutine.

4.1.336 Gap Height

Scope: Aria Material

Gap Height [{of|species|subindex} *Species*]= Constant [H = *h*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.337 Gap Height: Copied

Scope: Aria Material

Gap Height: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.338 Gap Height: User Plugin

Scope: Aria Material

Gap Height: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.339 Gas Phase Retardation

Scope: Aria Material

Gas Phase Retardation [{of|species|subindex} *Species*]= Constant [R = *r*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined

Summary Constant value

4.1.340 Gas Phase Retardation: Copied

Scope: Aria Material

Gas Phase Retardation: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.341 Gas Phase Retardation: User Plugin

Scope: Aria Material

Gas Phase Retardation: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.342 Glass Transition Temperature

Scope: Aria Material

Glass Transition Temperature [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.343 Glass Transition Temperature: Copied

Scope: Aria Material

Glass Transition Temperature: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.344 Glass Transition Temperature: User Plugin

Scope: Aria Material

Glass Transition Temperature: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.345 Glass Transition Temperature: Debenedeto

Scope: Aria Material

Glass Transition Temperature: Debenedeto [{of | species | subindex} *Species*]= Debenedeto [Extent_Species_Name = *extent_species_name* | A = *a* | Tg0 = *Tg0* | Tginfinity = *TgInfinity*]

Parameter	Value	Default
Species	string	undefined
extent_species_name	"string"	undefined
a	real	undefined
Tg0	real	undefined
TgInfinity	real	undefined

Summary deBenedeto model for the glass transition temperature based on the extent of polymerization, x, and material parameters $A, T_{g0}, T_{g\infty}$ $T_g = \frac{T_{g0}(1-x)+AxT_{g\infty}}{1-x+Ax}$.

4.1.346 Gnielinski Film Gradient Correction: Correlation

Scope: Aria Material

Gnielinski Film Gradient Correction: Correlation [{of | species | subindex} *Species*]= Correlation [Phase = *phase* | Gas_Exponent = *gas_exponent* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
phase	"string"	undefined
gas_exponent	real	undefined
bulk_node_interface	"string"	undefined

Summary Correction factor for difference between near-wall and bulk fluid state for Gnielinski pipe and annulus correlations.

4.1.347 Gravitational Constant: Correlation

Scope: Aria Material

Gravitational Constant: Correlation [{of | species | subindex} *Species*]= Correlation [G = *g* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
g	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Gravitational Constant

4.1.348 Hausen Entrance Effect Correction: Correlation

Scope: Aria Material

Hausen Entrance Effect Correction: Correlation [{of | species | subindex} *Species*]= Correlation [Max_Correction = *max_correction* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
max_correction	real	undefined
bulk_node_interface	"string"	undefined

Summary Correction factor for entrance effects for Gnielinski pipe and annulus correlations.

4.1.349 Hdiff Flux: Constant Cl

Scope: Aria Material

Hdiff Flux: Constant Cl [{of | species | subindex} *Species*]= Constant_Cl [Jx = *jx* | Jy = *jy* | Jz = *jz*]

Parameter	Value	Default
Species	string	undefined
jx	real	undefined
jy	real	undefined
jz	real	undefined

Summary ?

4.1.350 Heat Conduction: Generalized

Scope: Aria Material

Heat Conduction: Generalized [{of | species | subindex} *Species*]= Generalized [ ]

Parameter	Value	Default
Species	string	undefined

Summary Generalized model for heat conduction - can use either a scalar or tensor thermal conductivity.

4.1.351 Heat Conduction: Convected Enthalpy

Scope: Aria Material

Heat Conduction: Convected Enthalpy [{of | species | subindex} *Species*]= Convected_Enthalpy [ ]

Parameter	Value	Default
Species	string	undefined

Summary Convected enthalpy model for heat conduction

4.1.352 Heat Conduction: Phase Average

Scope: Aria Material

Heat Conduction: Phase Average [{of | species | subindex} *Species*]= Phase_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Phase average model for heat conduction

4.1.353 Heat Conduction: Thermoelectric

Scope: Aria Material

Heat Conduction: Thermoelectric [{of | species | subindex} *Species*]= Thermoelectric [ ]

Parameter	Value	Default
Species	string	undefined

Summary Thermoelectric model for heat conduction

4.1.354 Heat Conduction: Fouriers Law

Scope: Aria Material

Heat Conduction: Fouriers Law [{of | species | subindex} *Species*]= Fouriers_Law [ ]

Parameter	Value	Default
Species	string	undefined

Summary Fourier's Law heat conduction

4.1.355 Heat Conduction: Basic

Scope: Aria Material

Heat Conduction: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Basic heat conduction

4.1.356 Heat Conduction: Mass Diffusion Energy Transport

Scope: Aria Material

Heat Conduction: Mass Diffusion Energy Transport [{of | species | subindex} *Species*]= Mass_Diffusion_
[ ]

Parameter	Value	Default
Species	string	undefined

Summary Mass diffusion model for energy transport

4.1.357 Heat Conduction: Porous Diffusive Enthalpy Flux

Scope: Aria Material

`Heat Conduction: Porous Diffusive Enthalpy Flux [ {of|species|subindex} Species ]= Porous_Diffusive [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Porous diffusive enthalpy flux model

4.1.358 Heat Conduction: Porous Simplified Diffusive Enthalpy Flux

Scope: Aria Material

`Heat Conduction: Porous Simplified Diffusive Enthalpy Flux [ {of|species|subindex} Species ]= Porous_Simplified_Diffusive_Enthalpy_Flux [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Porous simplified diffusive enthalpy flux model

4.1.359 Heat Of Vaporization

Scope: Aria Material

`Heat Of Vaporization [ {of|species|subindex} Species ]= Constant [ Hv = hv ]`

Parameter	Value	Default
Species	string	undefined
hv	real	undefined

Summary Constant value

4.1.360 Heat Of Vaporization: Copied

Scope: Aria Material

`Heat Of Vaporization: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.361 Heat Of Vaporization: User Plugin

Scope: Aria Material

Heat Of Vaporization: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.362 Heat Transfer Coefficient

Scope: Aria Material

Heat Transfer Coefficient [{of|species|subindex} *Species*]= Constant [H = *h*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.363 Heat Transfer Coefficient: Copied

Scope: Aria Material

Heat Transfer Coefficient: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.364 Heat Transfer Coefficient: User Plugin

Scope: Aria Material

Heat Transfer Coefficient: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.365 Heat Transfer Coefficient: Calore User Sub

Scope: Aria Material

Heat Transfer Coefficient: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.366 Heat Transfer Coefficient: Aero

Scope: Aria Material

Heat Transfer Coefficient: Aero [{of|species|subindex} *Species*]= Aero [T_On = *t_on* |T_Off = *t_off* |Specific_Heat_Function = *specific_heat_function* |Conductivity_Function = *conductivity_function* |Viscosity_Function = *viscosity_function* |Offset_Direction = *offset_direction* |Coord_Offset = *coord_offset* |Cone_Length = *cone_length* |Gas_Constant = *gas_constant*]

Parameter	Value	Default
Species	string	undefined
t_on	real	undefined
t_off	real	undefined
specific_heat_function	"string"	undefined
conductivity_function	"string"	undefined
viscosity_function	"string"	undefined
offset_direction	"string"	undefined
coord_offset	"string"	undefined
cone_length	real	undefined
gas_constant	real	undefined

Summary Heat transfer coefficient for external flow about conical tipped body accounting for flow conditions including Mach number, altitude, fluid properties and local flow regime.

4.1.367 Heat Transfer Coefficient: Fortran

Scope: Aria Material

Heat Transfer Coefficient: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine heat transfer coefficient.

4.1.368 Heat Transfer Coefficient: Correlation

Scope: Aria Material

Heat Transfer Coefficient: Correlation [{of|species|subindex} *Species*]= Correlation [Rex = *Rex* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
Rex	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation based heat transfer coefficient

4.1.369 Hydraulic Diameter

Scope: Aria Material

Hydraulic Diameter [{of|species|subindex} *Species*]= Constant [D = *d*]

Parameter	Value	Default
Species	string	undefined
d	real	undefined

Summary Constant value

4.1.370 Hydraulic Diameter: Copied

Scope: Aria Material

Hydraulic Diameter: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.371 Hydraulic Diameter: User Plugin

Scope: Aria Material

Hydraulic Diameter: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.372 Hydraulic Diameter: Spatial User Function

Scope: Aria Material

Hydraulic Diameter: Spatial User Function [{of | species | subindex} *Species*]= Spatial_User_Function [Threshold = *threshold* | Multiplier = *multiplier* | Name = *name* | Bar = *bar* | X = *x*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined

Summary Spatially varying hydraulic diameter based on a coordinate dependent user function

4.1.373 Hydraulic Diameter: Correlation

Scope: Aria Material

Hydraulic Diameter: Correlation [{of | species | subindex} *Species*]= Correlation [D = *d* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
d	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation hydraulic diameter based on sectional area and wetted perimeter

4.1.374 Hydraulic Diameter: Correlation Spatial User Function

Scope: Aria Material

Hydraulic Diameter: Correlation Spatial User Function [{of | species | subindex} *Species*]= Correlation_Spatial_User_Function [Threshold = *threshold* | Multiplier = *multiplier* | Name = *name* | Bar = *bar* | X = *x* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined
bulk_node_interface	"string"	undefined

Summary Correlation spatially varying hydraulic diameter based on a coordinate dependent user function

4.1.375 Hydrogen Advection Velocity: Hdiff

Scope: Aria Material

Hydrogen Advection Velocity: Hdiff [{of|species|subindex} *Species*]= Hdiff [R = *r* | V_H = *v_h*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
v_h	real	undefined

Summary Advection velocity term for hydrogen transport

4.1.376 Hydrogen Diffusion Supg Tau: Shakib

Scope: Aria Material

Hydrogen Diffusion Supg Tau: Shakib [{of|species|subindex} *Species*]= Shakib [H = *h* |Use_Advection = *use_advection* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined
use_advection	integer	undefined
use_diffusion	integer	undefined
use_time	integer	undefined
multiplier	real	undefined

Summary ?

4.1.377 Induction Time Constant

Scope: Aria Material

Induction Time Constant [{of|species|subindex} *Species*]= Constant [Tau = *tau*]

Parameter	Value	Default
Species	string	undefined
tau	real	undefined

Summary Constant value

4.1.378 Induction Time Constant: Copied

Scope: Aria Material

Induction Time Constant: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.379 Induction Time Constant: User Plugin

Scope: Aria Material

Induction Time Constant: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.380 Initial Porosity

Scope: Aria Material

Initial Porosity [{of | species | subindex} *Species*]= Constant [Phi = *phi*]

Parameter	Value	Default
Species	string	undefined
phi	real	undefined

Summary Constant value

4.1.381 Initial Porosity: Copied

Scope: Aria Material

Initial Porosity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.382 Initial Porosity: User Plugin

Scope: Aria Material

Initial Porosity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.383 Intensity Diffusion: Spherical Harmonic

Scope: Aria Material

Intensity Diffusion: Spherical Harmonic [{of | species | subindex} *Species*]= Spherical_Harmonic [Spn_Order = *SPn_order* | Scaled = *scaled*]

Parameter	Value	Default
Species	string	undefined
SPn_order	integer	undefined
scaled	"string"	undefined

Summary Spherical harmonic intensity diffusion

4.1.384 Internal Energy

Scope: Aria Material

Internal Energy [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.385 Internal Energy: Copied

Scope: Aria Material

Internal Energy: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.386 Internal Energy: User Plugin

Scope: Aria Material

Internal Energy: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.387 Internal Energy: Gas Phase

Scope: Aria Material

Internal Energy: Gas Phase [{of | species | subindex} *Species*]= Gas_Phase [ ]

Parameter	Value	Default
Species	string	undefined

Summary Gas phase model for internal energy

4.1.388 Interphase Area

Scope: Aria Material

Interphase Area [{of | species | subindex} *Species*]= Constant [Sa = *sa*]

Parameter	Value	Default
Species	string	undefined
sa	real	undefined

Summary Constant value

4.1.389 Interphase Area: Copied

Scope: Aria Material

Interphase Area: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.390 Interphase Area: User Plugin

Scope: Aria Material

Interphase Area: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.391 Interphase Friction Coefficient: Wen Yu

Scope: Aria Material

Interphase Friction Coefficient: Wen Yu [{of|species|subindex} *Species*]= Wen_Yu [Diameter = *diameter*]

Parameter	Value	Default
Species	string	undefined
diameter	real	undefined

4.1.392 Intrinsic Permeability

Scope: Aria Material

Intrinsic Permeability [{of|species|subindex} *Species*]= Constant [Xx = *xx* |T11 = *t11* |Xy = *xy* |T12 = *t12* |Xz = *xz* |T13 = *t13* |Yx = *yx* |T21 = *t21* |Yy = *yy* |T22 = *t22* |Yz = *yz* |T23 = *t23* |Zx = *zx* |T31 = *t31* |Zy = *zy* |T32 = *t32* |Zz = *zz* |T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the intrinsic permeability

4.1.393 Intrinsic Permeability: Diagonal

Scope: Aria Material

Intrinsic Permeability: Diagonal [{of|species|subindex} *Species*]= Diagonal [Variable = *variable*]

Parameter	Value	Default
Species	string	undefined
variable	"string"	undefined

4.1.394 Intrinsic Permeability: Reordered Adagio

Scope: Aria Material

Intrinsic Permeability: Reordered Adagio [{of|species|subindex} *Species*]= Reordered_Adagio [Variable = *variable* |Restriction = *restriction*]

Parameter	Value	Default
Species	string	undefined
variable	"string"	undefined
restriction	"string"	undefined

4.1.395 Intrinsic Permeability: Log Reordered Adagio

Scope: Aria Material

Intrinsic Permeability: Log Reordered Adagio [{of|species|subindex} *Species*]= Log_Reordered_Adag
[Variable = *variable*]

Parameter	Value	Default
Species	`string`	`undefined`
variable	`"string"`	`undefined`

4.1.396 Intrinsic Permeability: Volume Average

Scope: Aria Material

`Intrinsic Permeability: Volume Average [ {of|species|subindex} Species ]= Volume_Average [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Average intrinsic permeability: sum (volume_fractions * permeabilities) / sum (volume_fractions)

4.1.397 Intrinsic Permeability: Young And Todd

Scope: Aria Material

`Intrinsic Permeability: Young And Todd [ {of|species|subindex} Species ]= Young_And_Todd [ Pore_Diameter = pore_diameter |Liquid_Phase_Name = Liquid_Phase_Name ]`

Parameter	Value	Default
Species	`string`	`undefined`
pore_diameter	`real`	`undefined`
Liquid_Phase_Name	`"string"`	`undefined`

Summary The Young and Todd permeability model described in SAND2010-0254. $\kappa = \frac{d^2}{32}\frac{\phi}{\tau^2}$

4.1.398 Intrinsic Permeability: Carmen Kozeny

Scope: Aria Material

`Intrinsic Permeability: Carmen Kozeny [ {of|species|subindex} Species ]= Carmen_Kozeny [ Sv = Sv |Min_Porosity = min_porosity |Liquid_Phase_Name = Liquid_Phase_Name ]`

Parameter	Value	Default
Species	`string`	`undefined`
Sv	`real`	`undefined`
min_porosity	`real`	`undefined`
Liquid_Phase_Name	`"string"`	`undefined`

Summary The Carmen-Kozeny model for permeability. $\kappa = \frac{1}{2\tau^2 S_v^2}\frac{\phi^3}{(1-\phi)^2}$, where S_v is the surface-to-volume ratio. This form is from B.R. Corrochano et al. Journal of Food Engineering 150 (2015) 106-116 and reduces to the traditional form if $S_v = 6/d$ and $\tau = \sqrt{(2.5)}$.

4.1.399 Intrinsic Permeability Scaling

Scope: Aria Material

Intrinsic Permeability Scaling [{of | species | subindex} *Species*]= Constant [S = *s*]

Parameter	Value	Default
Species	string	undefined
s	real	undefined

Summary Constant value

4.1.400 Intrinsic Permeability Scaling: Copied

Scope: Aria Material

Intrinsic Permeability Scaling: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.401 Intrinsic Permeability Scaling: User Plugin

Scope: Aria Material

Intrinsic Permeability Scaling: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.402 Intrinsic Permeability Scaling: Kozeny

Scope: Aria Material

Intrinsic Permeability Scaling: Kozeny [{of | species | subindex} *Species*]= Kozeny [Multiplier = *multiplier* | Phi_Ref = *phi_ref*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
phi_ref	real	undefined

Summary Intrinsic Permeability scaling function using Kozeny's model.

4.1.403 Intrinsic Permeability Scaling: T Exponent

Scope: Aria Material

Intrinsic Permeability Scaling: T Exponent [{of | species | subindex} *Species*]= T_Exponent [K_Ref = *k_ref* | T_Ref = *t_ref* | N = *n*]

Parameter	Value	Default
Species	string	undefined
k_ref	real	undefined
t_ref	real	undefined
n	real	undefined

Summary Intrinsic permeability scaling factor as a power of normalized temperature

4.1.404 Intrinsic Permeability Scaling: Forchheimer

Scope: Aria Material

Intrinsic Permeability Scaling: Forchheimer [{of | species | subindex} *Species*]= Forchheimer [Inertia_Coeff = *inertia_coeff* | Gx = *gx* | Gy = *gy* | Gz = *gz* | Xx = *xx* | Yy = *yy* | Zz = *zz* | Xy = *xy* | Xz = *xz* | Yz = *yz* | Yx = *yx* | Zx = *zx* | Zy = *zy* | Fluid_Phase = *fluid_phase*]

Parameter	Value	Default
Species	string	undefined
inertia_coeff	real	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined
xx	real	undefined
yy	real	undefined
zz	real	undefined
xy	real	undefined
xz	real	undefined
yz	real	undefined
yx	real	undefined
zx	real	undefined
zy	real	undefined
fluid_phase	"string"	undefined

Summary Intrinsic Permeability scaling for Forchheimer flow.

4.1.405 Inv Neutron Velocity

Scope: Aria Material

Inv Neutron Velocity [{of | species | subindex} *Species*]= Constant [Invnu = *InvNu*]

Parameter	Value	Default
Species	string	undefined
InvNu	real	undefined

Summary Constant value

4.1.406 Inv Neutron Velocity: Copied

Scope: Aria Material

```
Inv Neutron Velocity:  Copied [ {of | species | subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.407 Inv Neutron Velocity: User Plugin

Scope: Aria Material

```
Inv Neutron Velocity:  User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name
= Name  | magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.408 Irradiation

Scope: Aria Material

```
Irradiation [ {of | species | subindex} Species ]= Constant [ I = i ]
```

Parameter	Value	Default
Species	string	undefined
i	real	undefined

Summary Constant value

4.1.409 Irradiation: Copied

Scope: Aria Material

```
Irradiation:  Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.410 Irradiation: User Plugin

Scope: Aria Material

Irradiation: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.411 Irradiation: Calore User Sub

Scope: Aria Material

Irradiation: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Name = *name* | Type = *type* | Multiplier = *multiplier* | Material_Data_Block = *material_data_block* | Data = *data* | Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.412 Isobaric Compressibility

Scope: Aria Material

Isobaric Compressibility [{of | species | subindex} *Species*]= Constant [Beta = *beta*]

Parameter	Value	Default
Species	string	undefined
beta	real	undefined

Summary Constant value

4.1.413 Isobaric Compressibility: Copied

Scope: Aria Material

Isobaric Compressibility: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.414 Isobaric Compressibility: User Plugin

Scope: Aria Material

Isobaric Compressibility: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.415 K Factor

Scope: Aria Material

K Factor [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.416 K Factor: Copied

Scope: Aria Material

K Factor: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.417 K Factor: User Plugin

Scope: Aria Material

K Factor: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parameter	"string"	undefined

Summary Value from a user plugin

4.1.418 Kinematic Viscosity: Correlation

Scope: Aria Material

Kinematic Viscosity: Correlation [{of|species|subindex} *Species*]= Correlation [Nu = *nu* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
nu	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation kinematic viscosity

4.1.419 Lambda Brooks Corey

Scope: Aria Material

Lambda Brooks Corey [{of|species|subindex} *Species*]= Constant [Lambda = *lambda*]

Parameter	Value	Default
Species	string	undefined
lambda	real	undefined

Summary Constant value

4.1.420 Lambda Brooks Corey: Copied

Scope: Aria Material

Lambda Brooks Corey: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.421 Lambda Brooks Corey: User Plugin

Scope: Aria Material

Lambda Brooks Corey: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.422 Laminar Correlation Heat Transfer Coefficient: Correlation

Scope: Aria Material

Laminar Correlation Heat Transfer Coefficient: Correlation [{of | species | subindex} *Species*]= Correlation [Number = *number* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
number	integer	undefined
bulk_node_interface	"string"	undefined

Summary Correlation based heat transfer coefficient

4.1.423 Laser Power

Scope: Aria Material

Laser Power [{of | species | subindex} *Species*]= Constant [P = *p*]

Parameter	Value	Default
Species	string	undefined
p	real	undefined

Summary Constant value

4.1.424 Laser Power: Copied

Scope: Aria Material

Laser Power: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.425 Laser Power: User Plugin

Scope: Aria Material

`Laser Power: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.426 Latent Heat

Scope: Aria Material

`Latent Heat [ {of|species|subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	`string`	`undefined`
value	`real`	`undefined`

Summary Constant value

4.1.427 Latent Heat: Copied

Scope: Aria Material

`Latent Heat: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.428 Latent Heat: User Plugin

Scope: Aria Material

`Latent Heat: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.429 Latent Heat: Porous Phase Specific Average

Scope: Aria Material

Latent Heat: Porous Phase Specific Average [{of | species | subindex} *Species*]= Porous_Phase_Specifi
[Liquid_Phase_Name = *Liquid_Phase_Name* | Gas_Phase_Name = *Gas_Phase_Name*]

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined

Summary Latent heat for a multiphase material of two or three phases

4.1.430 Level Set

Scope: Aria Material

Level Set [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.431 Level Set: Copied

Scope: Aria Material

Level Set: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.432 Level Set: User Plugin

Scope: Aria Material

Level Set: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* |
magic_trailing_string_vector_param]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.433 Level Set Curvature

Scope: Aria Material

```
Level Set Curvature [ {of|species|subindex} Species ]= Constant [ Kappa = kappa ]
```

Parameter	Value	Default
Species	string	undefined
kappa	real	undefined

Summary Constant value

4.1.434 Level Set Curvature: Copied

Scope: Aria Material

```
Level Set Curvature: Copied [ {of|species|subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.435 Level Set Curvature: User Plugin

Scope: Aria Material

```
Level Set Curvature: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name
= Name |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.436 Level Set Diffusive Flux: Shock Capturing

Scope: Aria Material

```
Level Set Diffusive Flux: Shock Capturing [ {of|species|subindex} Species ]= Shock_Capturing
[ Multiplier = multiplier ]
```

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined

Summary Shock capturing operator for the level set equation

4.1.437 Level Set Diffusive Flux: Unit Gradient

Scope: Aria Material

Level Set Diffusive Flux: Unit Gradient [{of | species | subindex} *Species*]= Unit_Gradient [ ]

Parameter	Value	Default
Species	string	undefined

Summary Unit gradient model of level set diffusive flux

4.1.438 Level Set Diffusive Flux

Scope: Aria Material

Level Set Diffusive Flux [{of | species | subindex} *Species*]= Constant [Coeff = *coeff*]

Parameter	Value	Default
Species	string	undefined
coeff	real	undefined

Summary Constant model of level set diffusive flux

4.1.439 Level Set Heaviside

Scope: Aria Material

Level Set Heaviside [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.440 Level Set Heaviside: Copied

Scope: Aria Material

Level Set Heaviside: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.441 Level Set Heaviside: User Plugin

Scope: Aria Material

Level Set Heaviside: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parameter	"string"	undefined

Summary Value from a user plugin

4.1.442 Level Set Heaviside: Smooth

Scope: Aria Material

Level Set Heaviside: Smooth [{of | species | subindex} *Species*]= Smooth [Use_Crossing_Time = *use_crossing_time*]

Parameter	Value	Default
Species	string	undefined
use_crossing_time	integer	undefined

Summary Smooth level set Heaviside

4.1.443 Level Set Heaviside: Interpolated

Scope: Aria Material

Level Set Heaviside: Interpolated [{of | species | subindex} *Species*]= Interpolated [Epsilon = *epsilon*]

Parameter	Value	Default
Species	string	undefined
epsilon	real	undefined

Summary Interpolated level set Heaviside

4.1.444 Level Set Heaviside: Bf Interpolated

Scope: Aria Material

Level Set Heaviside: Bf Interpolated [{of | species | subindex} *Species*]= Bf_Interpolated [Use_Crossing_Time = *use_crossing_time*]

Parameter	Value	Default
Species	string	undefined
use_crossing_time	integer	undefined

Summary BF interpolated level set Heaviside

4.1.445 Level Set Heaviside: Sharp Analytic

Scope: Aria Material

Level Set Heaviside: Sharp Analytic [{of | species | subindex} *Species*]= Sharp_Analytic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Sharp analytic level set Heaviside

4.1.446 Level Set Width

Scope: Aria Material

Level Set Width [{of | species | subindex} *Species*]= Constant [Width = *width*]

Parameter	Value	Default
Species	string	undefined
width	real	undefined

Summary Specification of a constant width diffuse level set

4.1.447 Liquid Phase Retardation

Scope: Aria Material

Liquid Phase Retardation [{of | species | subindex} *Species*]= Constant [R = *r*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined

Summary Constant value

4.1.448 Liquid Phase Retardation: Copied

Scope: Aria Material

Liquid Phase Retardation: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.449 Liquid Phase Retardation: User Plugin

Scope: Aria Material

Liquid Phase Retardation: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.450 Lubrication Height

Scope: Aria Material

Lubrication Height [{of | species | subindex} *Species*]= Constant [H = *h*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.451 Lubrication Height: Copied

Scope: Aria Material

Lubrication Height: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.452 Lubrication Height: User Plugin

Scope: Aria Material

Lubrication Height: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.453 Lubrication Height: Combined

Scope: Aria Material

`Lubrication Height: Combined [ {of | species | subindex} Species ]= Combined [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Calculate the lubrication height by combining the upper and lower heights

4.1.454 Lubrication Height Lower

Scope: Aria Material

`Lubrication Height Lower [ {of | species | subindex} Species ]= Constant [ H_Lower = h_lower ]`

Parameter	Value	Default
Species	string	undefined
h_lower	real	undefined

Summary Constant value

4.1.455 Lubrication Height Lower: Copied

Scope: Aria Material

`Lubrication Height Lower: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.456 Lubrication Height Lower: User Plugin

Scope: Aria Material

`Lubrication Height Lower: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.457 Lubrication Height Upper

Scope: Aria Material

Lubrication Height Upper [{of|species|subindex} *Species*]= Constant [H_Upper = *h_upper*]

Parameter	Value	Default
Species	string	undefined
h_upper	real	undefined

Summary Constant value

4.1.458 Lubrication Height Upper: Copied

Scope: Aria Material

Lubrication Height Upper: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.459 Lubrication Height Upper: User Plugin

Scope: Aria Material

Lubrication Height Upper: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.460 Lubrication K

Scope: Aria Material

Lubrication K [{of|species|subindex} *Species*]= Constant [K = *k*]

Parameter	Value	Default
Species	string	undefined
k	real	undefined

Summary Constant value

4.1.461 Lubrication K: Copied

Scope: Aria Material

Lubrication K: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.462 Lubrication K: User Plugin

Scope: Aria Material

Lubrication K: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.463 Lubrication Velocity Lower

Scope: Aria Material

Lubrication Velocity Lower [{of | species | subindex} *Species*]= Constant [V_Lowerx = *v_lowerx* | V_Lowery = *v_lowery* | V_Lowerz = *v_lowerz*]

Parameter	Value	Default
Species	string	undefined
v_lowerx	real	undefined
v_lowery	real	undefined
v_lowerz	real	undefined

Summary Constant value

4.1.464 Lubrication Velocity Lower: User Plugin

Scope: Aria Material

Lubrication Velocity Lower: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.465 Lubrication Velocity Upper

Scope: Aria Material

Lubrication Velocity Upper [{of | species | subindex} *Species*]= Constant [V_Upperx = *v_upperx* | V_Uppery = *v_uppery* | V_Upperz = *v_upperz*]

Parameter	Value	Default
Species	string	undefined
v_upperx	real	undefined
v_uppery	real	undefined
v_upperz	real	undefined

Summary Constant value

4.1.466 Lubrication Velocity Upper: User Plugin

Scope: Aria Material

Lubrication Velocity Upper: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.467 Mach Number

Scope: Aria Material

Mach Number [{of | species | subindex} *Species*]= Constant [Ma = *Ma*]

Parameter	Value	Default
Species	string	undefined
Ma	real	undefined

Summary Constant value

4.1.468 Mach Number: Copied

Scope: Aria Material

Mach Number: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.469 Mach Number: User Plugin

Scope: Aria Material

Mach Number: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.470 Masking

Scope: Aria Material

Masking [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.471 Masking: Copied

Scope: Aria Material

Masking: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.472 Masking: User Plugin

Scope: Aria Material

Masking: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_*
]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.473 Mass Balance Advective Flux: Porous

Scope: Aria Material

`Mass Balance Advective Flux: Porous [ {of | species | subindex} *Species* ]= Porous [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass balance advection term from mass fraction and mass flux.

4.1.474 Mass Balance Advective Flux: Porous Upwind

Scope: Aria Material

`Mass Balance Advective Flux: Porous Upwind [ {of | species | subindex} *Species* ]= Porous_Upwind [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass balance advection term from mass fraction and mass flux.

4.1.475 Mass Balance Advective Flux: Single Phase

Scope: Aria Material

`Mass Balance Advective Flux: Single Phase [ {of | species | subindex} *Species* ]= Single_Phase [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass balance advection term from density and velocity

4.1.476 Mass Balance Diffusive Flux: Porous

Scope: Aria Material

`Mass Balance Diffusive Flux: Porous [ {of | species | subindex} *Species* ]= Porous [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass balance diffusive term from porosity, mass diffusivity, mass fraction and density.

4.1.477 Mass Balance Diffusive Flux: Nernst Planck

Scope: Aria Material

`Mass Balance Diffusive Flux: Nernst Planck [ {of|species|subindex} Species ]= Nernst_Planck [ F = f |R = r ]`

Parameter	Value	Default
Species	string	undefined
f	real	undefined
r	real	undefined

4.1.478 Mass Balance Diffusive Flux: Density

Scope: Aria Material

`Mass Balance Diffusive Flux: Density [ {of|species|subindex} Species ]= Density [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass balance diffusive term from porosity, mass diffusivity, mass fraction and density.

4.1.479 Mass Balance Diffusive Flux: Basic

Scope: Aria Material

`Mass Balance Diffusive Flux: Basic [ {of|species|subindex} Species ]= Basic [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass balance diffusive term from mass diffusivity, mass fraction and density.

4.1.480 Mass Diffusivity

Scope: Aria Material

`Mass Diffusivity [ {of|species|subindex} Species ]= Constant [ D = d ]`

Parameter	Value	Default
Species	string	undefined
d	real	undefined

Summary Constant value

4.1.481 Mass Diffusivity: Copied

Scope: Aria Material

`Mass Diffusivity: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.482 Mass Diffusivity: User Plugin

Scope: Aria Material

Mass Diffusivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.483 Mass Diffusivity: Cantera

Scope: Aria Material

Mass Diffusivity: Cantera [{of | species | subindex} *Species*]= Cantera [Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
poffset	real	undefined

Summary Cantera mass diffusivity

4.1.484 Mass Diffusivity: From Schmidt

Scope: Aria Material

Mass Diffusivity: From Schmidt [{of | species | subindex} *Species*]= From_Schmidt [ ]

Parameter	Value	Default
Species	string	undefined

Summary Mass diffusivity from a Schmidt number and viscosity

4.1.485 Mass Diffusivity: Air Water

Scope: Aria Material

Mass Diffusivity: Air Water [{of | species | subindex} *Species*]= Air_Water [Tortuosity = *tortuosity* | P_Ref = *p_ref* | D_Ref = *d_ref* | Exponent = *exponent*]

Parameter	Value	Default
Species	string	undefined
tortuosity	real	undefined
p_ref	real	undefined
d_ref	real	undefined
exponent	real	undefined

Summary Air/Water mass diffusivity

4.1.486 Mass Diffusivity: Arrhenius

Scope: Aria Material

Mass Diffusivity: Arrhenius [{of|species|subindex} *Species*]= Arrhenius [R = *r* |C = *c* |D = *d* |Q = *q*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
c	real	undefined
d	real	undefined
q	real	undefined

Summary Arrhenius mass diffusivity

4.1.487 Mass Diffusivity: Chapman Enskog

Scope: Aria Material

Mass Diffusivity: Chapman Enskog [{of|species|subindex} *Species*]= Chapman_Enskog [D0 = *d0* |P0 = *p0* |T0 = *t0*]

Parameter	Value	Default
Species	string	undefined
d0	real	undefined
p0	real	undefined
t0	real	undefined

Summary Mass diffusivity model for gases that scales with $T^{3/2}/P$ based on Chapman-Enskog theory.

4.1.488 Mass Flux: Darcy

Scope: Aria Material

Mass Flux: Darcy [{of|species|subindex} *Species*]= Darcy [ ]

Parameter	Value	Default
Species	string	undefined

4.1.489 Mass Fraction

Scope: Aria Material

`Mass Fraction [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.490 Mass Fraction: Copied

Scope: Aria Material

`Mass Fraction: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.491 Mass Fraction: User Plugin

Scope: Aria Material

`Mass Fraction: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.492 Mass Fraction: Fracbal

Scope: Aria Material

`Mass Fraction: Fracbal [ {of | species | subindex} Species ]= Fracbal [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass fraction of one species to yield a unity sum over all species

4.1.493 Mass Fraction: From Chemeq

Scope: Aria Material

`Mass Fraction: From Chemeq [ {of | species | subindex} Species ]= From_Chemeq [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mass fraction of one species extracted from ChemEq

4.1.494 Mass Fraction: From Density

Scope: Aria Material

`Mass Fraction: From Density [ {of | species | subindex} Species ]= From_Density [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Species density / overall density.

4.1.495 Mass Fraction Diffusive Flux: Basic

Scope: Aria Material

`Mass Fraction Diffusive Flux: Basic [ {of | species | subindex} Species ]= Basic [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Basic model of mass fraction diffusive flux

4.1.496 Mass Fraction Diffusive Flux: Energy Mass

Scope: Aria Material

`Mass Fraction Diffusive Flux: Energy Mass [ {of | species | subindex} Species ]= Energy_Mass [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Energy_Mass model of mass fraction diffusive flux

4.1.497 Material Data Block: Resource

Scope: Aria Material

`Material Data Block: Resource [ {of | species | subindex} Species ]= Resource [ Data_Block_Name = data_block_name ]`

Parameter	Value	Default
Species	string	undefined
data_block_name	"string"	undefined

Summary Resource material data block

4.1.498 Melt Temperature

Scope: Aria Material

```
Melt Temperature [ {of|species|subindex} Species ]= Constant [ Value = value ]
```

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.499 Melt Temperature: Copied

Scope: Aria Material

```
Melt Temperature:  Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.500 Melt Temperature: User Plugin

Scope: Aria Material

```
Melt Temperature:  User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name =
Name |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.501 Mesh Body Acceleration

Scope: Aria Material

```
Mesh Body Acceleration [ {of|species|subindex} Species ]= Constant [ Valuex = valuex |
Valuey = valuey |Valuez = valuez ]
```

Parameter	Value	Default
Species	string	undefined
valuex	real	undefined
valuey	real	undefined
valuez	real	undefined

Summary Constant value

4.1.502 Mesh Body Acceleration: User Plugin

Scope: Aria Material

Mesh Body Acceleration: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.503 Mesh Lambda

Scope: Aria Material

Mesh Lambda [{of|species|subindex} *Species*]= Constant [Lambda = *lambda*]

Parameter	Value	Default
Species	string	undefined
lambda	real	undefined

Summary Constant value

4.1.504 Mesh Lambda: Copied

Scope: Aria Material

Mesh Lambda: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.505 Mesh Lambda: User Plugin

Scope: Aria Material

Mesh Lambda: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.506 Mesh Lambda: Converted

Scope: Aria Material

Mesh Lambda: Converted [{of | species | subindex} *Species*]= Converted [ ]

Parameter	Value	Default
Species	string	undefined

Summary Lame coefficient computed from Poisson's ratio and elastic modulus

4.1.507 Mesh Lambda: Converted Plane Stress

Scope: Aria Material

Mesh Lambda: Converted Plane Stress [{of | species | subindex} *Species*]= Converted_Plane_Stress [ ]

Parameter	Value	Default
Species	string	undefined

Summary Lame coefficient applicable to plane stress conditions computed from Poisson's ratio and elastic modulus

4.1.508 Mesh Lambda: Elemental Volume

Scope: Aria Material

Mesh Lambda: Elemental Volume [{of | species | subindex} *Species*]= Elemental_Volume [ ]

Parameter	Value	Default
Species	string	undefined

Summary Lambda computed from elemental volume

4.1.509 Mesh Lambda: Inverse Element Volume

Scope: Aria Material

Mesh Lambda: Inverse Element Volume [{of | species | subindex} *Species*]= Inverse_Element_Volume [Lagged = *lagged*]

Parameter	Value	Default
Species	string	undefined
lagged	integer	undefined

Summary Lambda computed from inverse of element volume

4.1.510 Mesh Poissons Ratio

Scope: Aria Material

```
Mesh Poissons Ratio [ {of|species|subindex} Species ]= Constant [ Nu = nu ]
```

Parameter	Value	Default
Species	string	undefined
nu	real	undefined

Summary Constant value

4.1.511 Mesh Poissons Ratio: Copied

Scope: Aria Material

```
Mesh Poissons Ratio:  Copied [ {of|species|subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.512 Mesh Poissons Ratio: User Plugin

Scope: Aria Material

```
Mesh Poissons Ratio:  User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name
= Name  |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.513 Mesh Stress: Isothermal

Scope: Aria Material

```
Mesh Stress:  Isothermal [ {of|species|subindex} Species ]= Isothermal [ T = t  |T_Ref
= t_ref ]
```

Parameter	Value	Default
Species	string	undefined
t	real	undefined
t_ref	real	undefined

4.1.514 Mesh Stress: Linear Elastic

Scope: Aria Material

Mesh Stress: Linear Elastic [{of | species | subindex} *Species*]= Linear_Elastic [Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.515 Mesh Stress: Porous Effective

Scope: Aria Material

Mesh Stress: Porous Effective [{of | species | subindex} *Species*]= Porous_Effective [Pressure = *pressure* | Phase = *phase* | Biot = *biot*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined

Summary Apply the pressure as an isotropic mesh stress. Primarily used as an effective stress in porous flow problems.

4.1.516 Mesh Stress: Saturation Weighted Porous Effective

Scope: Aria Material

Mesh Stress: Saturation Weighted Porous Effective [{of | species | subindex} *Species*]= Saturation_We
[Pressure = *pressure* | Phase = *phase* | Biot = *biot*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined

Summary Apply the saturation-weighted pressure as an isotropic mesh stress. Primarily used as an effective stress in porous flow problems.

4.1.517 Mesh Stress: Mooney Rivlin

Scope: Aria Material

Mesh Stress: Mooney Rivlin [{of|species|subindex} *Species*]= Mooney_Rivlin [ ]

Parameter	Value	Default
Species	string	undefined

4.1.518 Mesh Stress: Neohookean Elastic

Scope: Aria Material

Mesh Stress: Neohookean Elastic [{of|species|subindex} *Species*]= Neohookean_Elastic [ ]

Parameter	Value	Default
Species	string	undefined

4.1.519 Mesh Stress: Nonlinear Elastic

Scope: Aria Material

Mesh Stress: Nonlinear Elastic [{of|species|subindex} *Species*]= Nonlinear_Elastic [ ]

Parameter	Value	Default
Species	string	undefined

4.1.520 Mesh Stress: Residual

Scope: Aria Material

Mesh Stress: Residual [{of|species|subindex} *Species*]= Residual [Sz = *sz* |Szz = *szz* |Sxz = *sxz* |Syz = *syz* |Sy = *sy* |Syy = *syy* |Sxy = *sxy* |Sx = *sx* |Sxx = *sxx*]

Parameter	Value	Default
Species	string	undefined
sz	real	undefined
szz	real	undefined
sxz	real	undefined
syz	real	undefined
sy	real	undefined
syy	real	undefined
sxy	real	undefined
sx	real	undefined
sxx	real	undefined

4.1.521 Mesh Stress: Thermal

Scope: Aria Material

Mesh Stress: Thermal [{of|species|subindex} *Species*]= Thermal [ ]

Parameter	Value	Default
Species	string	undefined

4.1.522 Mesh Stress: Electrode

Scope: Aria Material

Mesh Stress: Electrode [{of|species|subindex} *Species*]= Electrode [Electrodefile = *electrodeFile* |F = *f* |Kmolconversion = *kmolConversion* |Jconversion = *JConversion* |Meterconversion = *meterConversion* |Mincapacity = *minCapacity* |Deactivationvoltage = *deactivationVoltage* |Voltagevariablename = *voltageVariableName* |Abortaction = *abortAction* |Inactivesolidfraction = *inactiveSolidFraction* |Pref = *Pref* |Sens_Eps = *Sens_eps* |Sens_Order = *Sens_order* |Jam_Porosity = *jam_porosity* |Jam_Width = *jam_width*]

Parameter	Value	Default
Species	string	undefined
electrodeFile	"string"	undefined
f	real	undefined
kmolConversion	real	undefined
JConversion	real	undefined
meterConversion	real	undefined
minCapacity	real	undefined
deactivationVoltage	real	undefined
voltageVariableName	"string"	undefined
abortAction	"string"	undefined
inactiveSolidFraction	real	undefined
Pref	real	undefined
Sens_eps	real	undefined
Sens_order	"string"	undefined
jam_porosity	real	undefined
jam_width	real	undefined

4.1.523 Mesh Stress: Electrode Old

Scope: Aria Material

Mesh Stress: Electrode Old [{of|species|subindex} *Species*]= Electrode_Old [F = *f* |Electrodefile = *electrodeFile* |Kmolconversion = *kmolConversion* |Jconversion = *JConversion* |Meterconversion = *meterConversion* |Max_Sub_Timesteps = *max_sub_timesteps* |Base_Delta = *base_delta*]

Parameter	Value	Default
Species	string	undefined
f	real	undefined
electrodeFile	"string"	undefined
kmolConversion	real	undefined
JConversion	real	undefined
meterConversion	real	undefined
max_sub_timesteps	integer	undefined
base_delta	real	undefined

4.1.524 Mesh Stress: Species

Scope: Aria Material

Mesh Stress: Species [{of|species|subindex} *Species*]= Species [Sref = *sref* |Phase = *phase* |Species = *strain_species_name*]

Parameter	Value	Default
Species	string	undefined
sref	real	undefined
phase	"string"	undefined
strain_species_name	"string"	undefined

Summary A linear mechanical strain due to a species concentration

4.1.525 Mesh Stress: Species Transversely Isotropic

Scope: Aria Material

```
Mesh Stress: Species Transversely Isotropic [ {of|species|subindex} Species ]= Species_Transversely
[ Refval = refval |Beta_N = beta_n |Beta_T = beta_t |N_X = n_x |N_Y = n_y |N_Z = n_z |
Phase = phase ]
```

Parameter	Value	Default
Species	string	undefined
refval	real	undefined
beta_n	real	undefined
beta_t	real	undefined
n_x	real	undefined
n_y	real	undefined
n_z	real	undefined
phase	"string"	undefined

Summary A linear mechanical strain that is transversely isotropic due to a species concentration

4.1.526 Mesh Stress: Species Anisotropic

Scope: Aria Material

```
Mesh Stress: Species Anisotropic [ {of|species|subindex} Species ]= Species_Anisotropic
[ Sref = sref |Phase = phase ]
```

Parameter	Value	Default
Species	string	undefined
sref	real	undefined
phase	"string"	undefined

Summary Anisotropic linear mechanical strain due to a species concentration

4.1.527 Mesh Stress: Lame

Scope: Aria Material

```
Mesh Stress: Lame [ {of|species|subindex} Species ]= Lame [ Model = model ]
```

Parameter	Value	Default
Species	string	undefined
model	"string"	undefined

4.1.528 Mesh Two Mu

Scope: Aria Material

`Mesh Two Mu [ {of | species | subindex} Species ]= Constant [ Two_Mu = two_mu ]`

Parameter	Value	Default
Species	string	undefined
two_mu	real	undefined

Summary Constant value

4.1.529 Mesh Two Mu: Copied

Scope: Aria Material

`Mesh Two Mu: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.530 Mesh Two Mu: User Plugin

Scope: Aria Material

`Mesh Two Mu: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.531 Mesh Two Mu: Converted

Scope: Aria Material

`Mesh Two Mu: Converted [ {of | species | subindex} Species ]= Converted [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Lame coefficient computed from Poisson's ratio and elastic modulus

4.1.532 Mesh Two Mu: Elemental Volume

Scope: Aria Material

Mesh Two Mu: Elemental Volume [{of | species | subindex} *Species*]= Elemental_Volume [ ]

Parameter	Value	Default
Species	string	undefined

Summary 2 mu computed from elemental volume

4.1.533 Mesh Two Mu: Inverse Element Volume

Scope: Aria Material

Mesh Two Mu: Inverse Element Volume [{of | species | subindex} *Species*]= Inverse_Element_Volume [Lagged = *lagged*]

Parameter	Value	Default
Species	string	undefined
lagged	integer	undefined

Summary 2 mu computed from inverse of element volume

4.1.534 Mesh Two Mu: Faux Plasticity

Scope: Aria Material

Mesh Two Mu: Faux Plasticity [{of | species | subindex} *Species*]= Faux_Plasticity [E = *e* | Yield_Stress = *yield_stress* | Hard = *hard* | Min_Stress_Multiplier = *min_stress_multiplier*]

Parameter	Value	Default
Species	string	undefined
e	real	undefined
yield_stress	real	undefined
hard	"string"	undefined
min_stress_multiplier	real	undefined

4.1.535 Mesh Youngs Modulus

Scope: Aria Material

Mesh Youngs Modulus [{of | species | subindex} *Species*]= Constant [E = *e*]

Parameter	Value	Default
Species	string	undefined
e	real	undefined

Summary Constant value

4.1.536 Mesh Youngs Modulus: Copied

Scope: Aria Material

Mesh Youngs Modulus: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.537 Mesh Youngs Modulus: User Plugin

Scope: Aria Material

Mesh Youngs Modulus: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.538 Mesh Youngs Modulus: Melting Jamming Porous

Scope: Aria Material

Mesh Youngs Modulus: Melting Jamming Porous [{of | species | subindex} *Species*]= Melting_Jamming_Por [Jam_Porosity = *jam_porosity* | Jam_Width = *jam_width* | Solid_Modulus = *solid_modulus* | Melted_Modulus = *melted_modulus* | Jammed_Modulus = *jammed_modulus* | Melt_Dt = *melt_DT*]

Parameter	Value	Default
Species	string	undefined
jam_porosity	real	undefined
jam_width	real	undefined
solid_modulus	real	undefined
melted_modulus	real	undefined
jammed_modulus	real	undefined
melt_DT	real	undefined

4.1.539 Mesh Youngs Modulus: Faux Plasticity

Scope: Aria Material

Mesh Youngs Modulus: Faux Plasticity [{of | species | subindex} *Species*]= Faux_Plasticity [E = *e* | Yield_Stress = *yield_stress* | Hard = *hard* | Min_Stress_Multiplier = *min_stress_multiplier*]

Parameter	Value	Default
Species	string	undefined
e	real	undefined
yield_stress	real	undefined
hard	"string"	undefined
min_stress_multiplier	real	undefined

4.1.540 Mixture Fraction

Scope: Aria Material

```
Mixture Fraction [ {of | species | subindex} Species ]= Constant [ Value = value ]
```

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.541 Mixture Fraction: Copied

Scope: Aria Material

```
Mixture Fraction: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.542 Mixture Fraction: User Plugin

Scope: Aria Material

```
Mixture Fraction: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name =
Name | magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.543 Mixture Fraction Diffusive Flux: Basic

Scope: Aria Material

```
Mixture Fraction Diffusive Flux: Basic [ {of | species | subindex} Species ]= Basic [ ]
```

Parameter	Value	Default
Species	string	undefined

Summary Basic model of mixture fraction diffusive flux

4.1.544 Mixture Fraction Diffusivity

Scope: Aria Material

Mixture Fraction Diffusivity [{of | species | subindex} *Species*]= Constant [Dz = *Dz*]

Parameter	Value	Default
Species	string	undefined
Dz	real	undefined

Summary Constant value

4.1.545 Mixture Fraction Diffusivity: Copied

Scope: Aria Material

Mixture Fraction Diffusivity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.546 Mixture Fraction Diffusivity: User Plugin

Scope: Aria Material

Mixture Fraction Diffusivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.547 Mixture Fraction Diffusivity: From Schmidt

Scope: Aria Material

Mixture Fraction Diffusivity: From Schmidt [{of | species | subindex} *Species*]= From_Schmidt [ ]

Parameter	Value	Default
Species	string	undefined

Summary Mixture fraction diffusivity from Schmidt number and viscosity

4.1.548 Mixture Mass Diffusivity

Scope: Aria Material

Mixture Mass Diffusivity [{of | species | subindex} *Species*]= Constant [Dmix = *Dmix*]

Parameter	Value	Default
Species	string	undefined
Dmix	real	undefined

Summary Constant value

4.1.549 Mixture Mass Diffusivity: Copied

Scope: Aria Material

Mixture Mass Diffusivity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.550 Mixture Mass Diffusivity: User Plugin

Scope: Aria Material

Mixture Mass Diffusivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.551 Mixture Mass Diffusivity: Mixture Average

Scope: Aria Material

Mixture Mass Diffusivity: Mixture Average [{of | species | subindex} *Species*]= Mixture_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Linear mixture-averaged mass diffusivity

4.1.552 Mixture Molecular Weight

Scope: Aria Material

Mixture Molecular Weight [{of | species | subindex} *Species*]= Constant [Mmix = *Mmix*]

Parameter	Value	Default
Species	string	undefined
Mmix	real	undefined

Summary Constant value

4.1.553 Mixture Molecular Weight: Copied

Scope: Aria Material

Mixture Molecular Weight: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.554 Mixture Molecular Weight: User Plugin

Scope: Aria Material

Mixture Molecular Weight: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.555 Mixture Molecular Weight: Cantera

Scope: Aria Material

Mixture Molecular Weight: Cantera [{of | species | subindex} *Species*]= Cantera [Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
poffset	real	undefined

Summary Cantera mixture molecular weight

4.1.556 Mixture Molecular Weight: Mass Average

Scope: Aria Material

```
Mixture Molecular Weight:  Mass Average [ {of | species | subindex} Species ]= Mass_Average
[ ]
```

Parameter	Value	Default
Species	string	undefined

Summary Mass averaged mixture molecular weight

4.1.557 Mixture Molecular Weight: Mole Average

Scope: Aria Material

```
Mixture Molecular Weight:  Mole Average [ {of | species | subindex} Species ]= Mole_Average
[ Species_Variable = species_variable ]
```

Parameter	Value	Default
Species	string	undefined
species_ variable	"string"	undefined

Summary Mole averaged mixture molecular weight

4.1.558 Molecular Weight

Scope: Aria Material

```
Molecular Weight [ {of | species | subindex} Species ]= Constant [ M = m ]
```

Parameter	Value	Default
Species	string	undefined
m	real	undefined

Summary Constant value

4.1.559 Molecular Weight: Copied

Scope: Aria Material

```
Molecular Weight:  Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.560 Molecular Weight: User Plugin

Scope: Aria Material

`Molecular Weight: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.561 Molecular Weight: Cantera

Scope: Aria Material

`Molecular Weight: Cantera [ {of | species | subindex} Species ]= Cantera [ Poffset = poffset ]`

Parameter	Value	Default
Species	`string`	`undefined`
poffset	`real`	`undefined`

Summary Cantera molecular weight

4.1.562 Molecular Weight: Dynamic Species

Scope: Aria Material

`Molecular Weight: Dynamic Species [ {of | species | subindex} Species ]= Dynamic_Species [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Calculate the molecular weight dynamically from density/molar concentration.

4.1.563 Momentum Face Stabilization Scaling: Default

Scope: Aria Material

Summary Default face stabilization scaling for momentum equation

4.1.564 Momentum Stress: Bad Sensitivity

Scope: Aria Material

Momentum Stress: Bad Sensitivity [{of|species|subindex} *Species*]= Bad_Sensitivity [ ]

Parameter	Value	Default
Species	string	undefined

4.1.565 Momentum Stress: Balanced Force

Scope: Aria Material

Momentum Stress: Balanced Force [{of|species|subindex} *Species*]= Balanced_Force [ ]

Parameter	Value	Default
Species	string	undefined

4.1.566 Momentum Stress: Grad Div

Scope: Aria Material

Momentum Stress: Grad Div [{of|species|subindex} *Species*]= Grad_Div [Pressure = *pressure* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
multiplier	real	undefined

4.1.567 Momentum Stress: Formal Newtonian

Scope: Aria Material

Momentum Stress: Formal Newtonian [{of|species|subindex} *Species*]= Formal_Newtonian [Pressure = *pressure* |Phase = *phase* |Biot = *biot* |Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined
reference_frame	"string"	undefined

4.1.568 Momentum Stress: Incompressible Newtonian

Scope: Aria Material

Momentum Stress: Incompressible Newtonian [{of|species|subindex} *Species*]= Incompressible_Newtoni [Pressure = *pressure* |Phase = *phase* |Biot = *biot* |Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined
reference_frame	"string"	undefined

4.1.569 Momentum Stress: Suspension

Scope: Aria Material

Momentum Stress: Suspension [{of | species | subindex} *Species*]= Suspension [Pressure = *pressure* |Phase = *phase* |Biot = *biot* |Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined
reference_frame	"string"	undefined

4.1.570 Momentum Stress: Ls Capillary

Scope: Aria Material

Momentum Stress: Ls Capillary [{of | species | subindex} *Species*]= Ls_Capillary [ ]

Parameter	Value	Default
Species	string	undefined

4.1.571 Momentum Stress: Ls Implicit Capillary

Scope: Aria Material

Momentum Stress: Ls Implicit Capillary [{of | species | subindex} *Species*]= Ls_Implicit_Capillary [ ]

Parameter	Value	Default
Species	string	undefined

4.1.572 Momentum Stress: Ls Implicit Normal Capillary

Scope: Aria Material

Momentum Stress: Ls Implicit Normal Capillary [{of | species | subindex} *Species*]= Ls_Implicit_Normal [Mu0 = *mu0*]

Parameter	Value	Default
Species	string	undefined
mu0	real	undefined

4.1.573 Momentum Stress: Newtonian Dilational

Scope: Aria Material

Momentum Stress: Newtonian Dilational [{of | species | subindex} *Species*]= Newtonian_Dilational [Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.574 Momentum Stress: Newtonian Viscous

Scope: Aria Material

Momentum Stress: Newtonian Viscous [{of | species | subindex} *Species*]= Newtonian_Viscous [Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.575 Momentum Stress: Newtonian Pressure

Scope: Aria Material

Momentum Stress: Newtonian Pressure [{of | species | subindex} *Species*]= Newtonian_Pressure [Pressure = *pressure* | Phase = *phase* | Biot = *biot*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined

4.1.576 Momentum Stress: Maxwell

Scope: Aria Material

Momentum Stress: Maxwell [{of | species | subindex} *Species*]= Maxwell [ ]

Parameter	Value	Default
Species	string	undefined

Summary The gradient of the Maxwell stress tensor is the force on a fluid due to an electrical field and is non-zero when there is either a net charge density or a non-uniform electric permittivity. $\boldsymbol{\nabla} \cdot \mathbf{M} = \rho_{charge}\mathbf{E} - \frac{1}{2}(\mathbf{E} \cdot \mathbf{E})\boldsymbol{\nabla}\epsilon$

4.1.577 Neutron

Scope: Aria Material

Neutron [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.578 Neutron: Copied

Scope: Aria Material

Neutron: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.579 Neutron: User Plugin

Scope: Aria Material

Neutron: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_*
]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parser	"string"	undefined

Summary Value from a user plugin

4.1.580 Neutron Diffusion: Ficks Law

Scope: Aria Material

Neutron Diffusion: Ficks Law [{of | species | subindex} *Species*]= Ficks_Law [ ]

Parameter	Value	Default
Species	string	undefined

Summary Fick's Law neutron diffusion

4.1.581 Neutron Diffusion: Basic

Scope: Aria Material

Neutron Diffusion: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Basic neutron diffusion

4.1.582 Neutron Diffusivity

Scope: Aria Material

```
Neutron Diffusivity [ {of|species|subindex} Species ]= Constant [ D = d ]
```

Parameter	Value	Default
Species	string	undefined
d	real	undefined

Summary Constant value

4.1.583 Neutron Diffusivity: Copied

Scope: Aria Material

```
Neutron Diffusivity: Copied [ {of|species|subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.584 Neutron Diffusivity: User Plugin

Scope: Aria Material

```
Neutron Diffusivity: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name
= Name |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.585 Neutron Diffusivity: Five Ten

Scope: Aria Material

```
Neutron Diffusivity: Five Ten [ {of|species|subindex} Species ]= Five_Ten [ ]
```

Parameter	Value	Default
Species	string	undefined

Summary Five ten neutron diffusivity

4.1.586 Nonwetting Phase Retardation

Scope: Aria Material

Nonwetting Phase Retardation [{of | species | subindex} *Species*]= Constant [R = *r*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined

Summary Constant value

4.1.587 Nonwetting Phase Retardation: Copied

Scope: Aria Material

Nonwetting Phase Retardation: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.588 Nonwetting Phase Retardation: User Plugin

Scope: Aria Material

Nonwetting Phase Retardation: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.589 Partial Enclosure Area

Scope: Aria Material

Partial Enclosure Area [{of | species | subindex} *Species*]= Constant [A = *a*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined

Summary Constant value

4.1.590 Partial Enclosure Area: Copied

Scope: Aria Material

Partial Enclosure Area: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.591 Partial Enclosure Area: User Plugin

Scope: Aria Material

Partial Enclosure Area: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.592 Partial Enclosure Area: Calore User Sub

Scope: Aria Material

Partial Enclosure Area: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.593 Partial Enclosure Emissivity

Scope: Aria Material

Partial Enclosure Emissivity [{of | species | subindex} *Species*]= Constant [E = *e*]

Parameter	Value	Default
Species	string	undefined
e	real	undefined

Summary Constant value

4.1.594 Partial Enclosure Emissivity: Copied

Scope: Aria Material

Partial Enclosure Emissivity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.595 Partial Enclosure Emissivity: User Plugin

Scope: Aria Material

Partial Enclosure Emissivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.596 Partial Enclosure Emissivity: Calore User Sub

Scope: Aria Material

Partial Enclosure Emissivity: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Name = *name* | Type = *type* | Multiplier = *multiplier* | Material_Data_Block = *material_data_block* | Data = *data* | Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.597 Partial Enclosure Temperature

Scope: Aria Material

Partial Enclosure Temperature [{of | species | subindex} *Species*]= Constant [T = *t*]

Parameter	Value	Default
Species	string	undefined
t	real	undefined

Summary Constant value

4.1.598 Partial Enclosure Temperature: Copied

Scope: Aria Material

Partial Enclosure Temperature: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.599 Partial Enclosure Temperature: User Plugin

Scope: Aria Material

Partial Enclosure Temperature: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.600 Partial Enclosure Temperature: Calore User Sub

Scope: Aria Material

Partial Enclosure Temperature: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.601 Partial Molar Volume

Scope: Aria Material

Partial Molar Volume [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.602 Partial Molar Volume: Copied

Scope: Aria Material

Partial Molar Volume: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.603 Partial Molar Volume: User Plugin

Scope: Aria Material

Partial Molar Volume: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.604 Peltier Coefficient

Scope: Aria Material

Peltier Coefficient [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.605 Peltier Coefficient: Copied

Scope: Aria Material

Peltier Coefficient: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.606 Peltier Coefficient: User Plugin

Scope: Aria Material

Peltier Coefficient: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.607 Phase Change Specific Heat

Scope: Aria Material

Phase Change Specific Heat [{of | species | subindex} *Species*]= Constant [Cp = *cp*]

Parameter	Value	Default
Species	string	undefined
cp	real	undefined

Summary Constant value

4.1.608 Phase Change Specific Heat: Copied

Scope: Aria Material

Phase Change Specific Heat: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.609 Phase Change Specific Heat: User Plugin

Scope: Aria Material

Phase Change Specific Heat: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.610 Phase Change Specific Heat: Calore User Sub

Scope: Aria Material

Phase Change Specific Heat: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Name = *name* | Type = *type* | Multiplier = *multiplier* | Material_Data_Block = *material_data_block* | Data = *data* | Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.611 Phase Change Specific Heat: Curing Foam

Scope: Aria Material

`Phase Change Specific Heat: Curing Foam [ {of|species|subindex} Species ]= Curing_Foam [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Phase change specific heat for a curing foam

4.1.612 Phi

Scope: Aria Material

`Phi [ {of|species|subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.613 Phi: Copied

Scope: Aria Material

`Phi: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.614 Phi: User Plugin

Scope: Aria Material

`Phi: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_str ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parser	"string"	undefined

Summary Value from a user plugin

4.1.615 Polymerization Shift Factor

Scope: Aria Material

Polymerization Shift Factor [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.616 Polymerization Shift Factor: Copied

Scope: Aria Material

Polymerization Shift Factor: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.617 Polymerization Shift Factor: User Plugin

Scope: Aria Material

Polymerization Shift Factor: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.618 Polymerization Shift Factor: Wlf

Scope: Aria Material

Polymerization Shift Factor: Wlf [{of | species | subindex} *Species*]= Wlf [C1 = *c1* | C2 = *c2*]

Parameter	Value	Default
Species	string	undefined
c1	real	undefined
c2	real	undefined

4.1.619 Porosity

Scope: Aria Material

`Porosity [ {of | species | subindex} Species ]= Constant [ Phi = phi ]`

Parameter	Value	Default
Species	string	undefined
phi	real	undefined

Summary Constant value

4.1.620 Porosity: Deforming

Scope: Aria Material

`Porosity: Deforming [ {of | species | subindex} Species ]= Deforming [ Phi_Init = phi_init ]`

Parameter	Value	Default
Species	string	undefined
phi_init	real	undefined

Summary Porosity accounting for deforming media

4.1.621 Porosity: Mesh Deforming

Scope: Aria Material

`Porosity: Mesh Deforming [ {of | species | subindex} Species ]= Mesh_Deforming [ Biot = biot ]`

Parameter	Value	Default
Species	string	undefined
biot	real	undefined

Summary Rock compressibility linearized model of porosity for deforming media

4.1.622 Porosity: Solid Deforming

Scope: Aria Material

`Porosity: Solid Deforming [ {of | species | subindex} Species ]= Solid_Deforming [ Biot = biot ]`

Parameter	Value	Default
Species	string	undefined
biot	real	undefined

Summary Rock compressibility linearized model of porosity for deforming media

4.1.623 Porosity: Rock Compressible

Scope: Aria Material

Porosity: Rock Compressible [{of | species | subindex} *Species*]= Rock_Compressible [Cr = *cr* | Ref_Porosity = *ref_porosity* | Ref_Pressure = *ref_pressure*]

Parameter	Value	Default
Species	string	undefined
cr	real	undefined
ref_porosity	real	undefined
ref_pressure	real	undefined

Summary Rock compressibility linearized model of porosity

4.1.624 Porosity: Deformable Rock Compressible

Scope: Aria Material

Porosity: Deformable Rock Compressible [{of | species | subindex} *Species*]= Deformable_Rock_Compressi [Cr = *cr* | Ref_Porosity = *ref_porosity* | Ref_Pressure = *ref_pressure* | Biot = *biot*]

Parameter	Value	Default
Species	string	undefined
cr	real	undefined
ref_porosity	real	undefined
ref_pressure	real	undefined
biot	real	undefined

Summary Rock compressibility linearized model of porosity for deforming media

4.1.625 Porosity: Coussy

Scope: Aria Material

Porosity: Coussy [{of | species | subindex} *Species*]= Coussy [M_Inv = *m_inv* | Biot = *biot* | Ref_Porosity = *ref_porosity* | Ref_Pressure = *ref_pressure*]

Parameter	Value	Default
Species	string	undefined
m_inv	real	undefined
biot	real	undefined
ref_porosity	real	undefined
ref_pressure	real	undefined

Summary Coussy's model of porosity for deforming media (based on pore pressure and div(u)

4.1.626 Porosity: From Density Ratio

Scope: Aria Material

Porosity: From Density Ratio [{of | species | subindex} *Species*]= From_Density_Ratio [Solid_Density = *solid_density*]

Parameter	Value	Default
Species	string	undefined
solid_density	real	undefined

Summary Porosity computed as one minus the density ratio between solved for solid-phase density and solid density of non-porous material.

4.1.627 Porosity: One Minus Volume Fractions

Scope: Aria Material

Porosity: One Minus Volume Fractions [{of | species | subindex} *Species*]= One_Minus_Volume_Fractions [Phi_Min = *phi_min*]

Parameter	Value	Default
Species	string	undefined
phi_min	real	undefined

Summary Porosity computed directly from volume fractions of species.

4.1.628 Porosity: Constant From Electrode Object Old

Scope: Aria Material

Porosity: Constant From Electrode Object Old [{of | species | subindex} *Species*]= Constant_From_Elec [Electrodefile = *electrodeFile*]

Parameter	Value	Default
Species	string	undefined
electrodeFile	"string"	undefined

Summary Constant porosity extracted from Cantera thermal battery electrode object.

4.1.629 Porosity: Electrode Object

Scope: Aria Material

Porosity: Electrode Object [{of | species | subindex} *Species*]= Electrode_Object [Electrodefile = *electrodeFile* | F = *f* | Kmolconversion = *kmolConversion* | Jconversion = *JConversion* | Meterconversion = *meterConversion* | Mincapacity = *minCapacity* | Deactivationvoltage = *deactivationVoltage* | Voltagevariablename = *voltageVariableName* | Abortaction = *abortAction* | Inactivesolidfraction = *inactiveSolidFraction* | Pref = *Pref* | Sens_Eps = *Sens_eps* | Sens_Order = *Sens_order* | Min_Porosity = *min_porosity*]

Parameter	Value	Default
Species	string	undefined
electrodeFile	"string"	undefined
f	real	undefined
kmolConversion	real	undefined
JConversion	real	undefined
meterConversion	real	undefined
minCapacity	real	undefined
deactivationVoltage	real	undefined
voltageVariableName	"string"	undefined
abortAction	"string"	undefined
inactiveSolidFraction	real	undefined
Pref	real	undefined
Sens_eps	real	undefined
Sens_order	"string"	undefined
min_porosity	real	undefined

Summary Variable porosity extracted from Cantera thermal battery electrode object.

4.1.630 Porosity: From Volume Fraction Gas

Scope: Aria Material

Porosity: From Volume Fraction Gas [{of|species|subindex} *Species*]= From_Volume_Fraction_Gas [ ]

Parameter	Value	Default
Species	string	undefined

Summary Porosity from the defined volume fraction of gas

4.1.631 Porosity: Modified Kozeny

Scope: Aria Material

Porosity: Modified Kozeny [{of|species|subindex} *Species*]= Modified_Kozeny [C = *c* |B1 = *b1* |B2 = *b2* |L_Bed = *L_bed* |R_Bed = *R_bed* |Nx = *nx*]

Parameter	Value	Default
Species	string	undefined
c	real	undefined
b1	real	undefined
b2	real	undefined
L_bed	real	undefined
R_bed	real	undefined
nx	real	undefined

Summary Porosity of a cylindrical particle bed of domain of length L and radius R using experimental model parameters c, $b1$, $b2$ and nx. For 2D $r = x$ and 3D $r = \sqrt{x^2 + z^2}$ with $\bar{r} = r/R$, $\bar{F}1 = 1-(1-b1)/2$, $\bar{F}2 = b2+(1-b2)/2$, $F1 = 1-(1-b1)y/L$, $F2 = b2+(1-b2)(1-\bar{r}^{nx})$, then the porosity $\phi = cF1F2/\bar{F}1\bar{F}2$.

4.1.632 Porosity Def

Scope: Aria Material

`Porosity Def [ {of | species | subindex} Species ]= Constant [ Phi = phi ]`

Parameter	Value	Default
Species	`string`	`undefined`
phi	`real`	`undefined`

Summary Constant value

4.1.633 Porosity Def: Copied

Scope: Aria Material

`Porosity Def: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.634 Porosity Def: User Plugin

Scope: Aria Material

`Porosity Def: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.635 Porosity Def: Deforming

Scope: Aria Material

`Porosity Def: Deforming [ {of | species | subindex} Species ]= Deforming [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Deforming model for porosity def

4.1.636 Porous Flow System: Single Phase

Scope: Aria Material

Porous Flow System: Single Phase [{of|species|subindex} *Species*]= Single_Phase [Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

4.1.637 Porous Flow System: Mixed Single Phase

Scope: Aria Material

Porous Flow System: Mixed Single Phase [{of|species|subindex} *Species*]= Mixed_Single_Phase [Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

4.1.638 Porous Flow System: Two Phase Immiscible

Scope: Aria Material

Porous Flow System: Two Phase Immiscible [{of|species|subindex} *Species*]= Two_Phase_Immiscible [Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

4.1.639 Porous Flow System: Air Water

Scope: Aria Material

Porous Flow System: Air Water [{of|species|subindex} *Species*]= Air_Water [Gx = *gx* | Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

4.1.640 Porous Flow System: Co2 Brine Salt

Scope: Aria Material

Porous Flow System: Co2 Brine Salt [{of|species|subindex} *Species*]= Co2_Brine_Salt [X_Sat_Co2_File_Name = *x_sat_co2_file_name* |Rho_G_File_Name = *rho_g_file_name* |Rho_Co2_File_Name = *rho_co2_file_name* |Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
x_sat_co2_file_name	"string"	undefined
rho_g_file_name	"string"	undefined
rho_co2_file_name	"string"	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

4.1.641 Pp: Mesh Von Mises

Scope: Aria Material

Pp: Mesh Von Mises [{of|species|subindex} *Species*]= Mesh_Von_Mises [ ]

Parameter	Value	Default
Species	string	undefined

Summary Mesh Von Mises model

4.1.642 Pp: Solid Von Mises

Scope: Aria Material

Pp: Solid Von Mises [{of|species|subindex} *Species*]= Solid_Von_Mises [ ]

Parameter	Value	Default
Species	string	undefined

Summary Solid Von Mises model

4.1.643 Pp: Fluid Traction

Scope: Aria Material

Pp: Fluid Traction [{of|species|subindex} *Species*]= Fluid_Traction [ ]

Parameter	Value	Default
Species	string	undefined

Summary Fluid Traction model

4.1.644 Pp: Viscous Traction

Scope: Aria Material

`Pp: Viscous Traction [ {of|species|subindex} Species ]= Viscous_Traction [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Viscous Traction model - like Surface_Traction but omits the pressure contribution

4.1.645 Pp: Mesh Traction

Scope: Aria Material

`Pp: Mesh Traction [ {of|species|subindex} Species ]= Mesh_Traction [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Mesh Traction model

4.1.646 Pp: Solid Traction

Scope: Aria Material

`Pp: Solid Traction [ {of|species|subindex} Species ]= Solid_Traction [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Solid Traction model

4.1.647 Pp: Cvfem Yplus

Scope: Aria Material

`Pp: Cvfem Yplus [ {of|species|subindex} Species ]= Cvfem_Yplus [ ]`

Parameter	Value	Default
Species	string	undefined

Summary CVFEM YPLUS model

4.1.648 Pp: Yplus

Scope: Aria Material

`Pp: Yplus [ {of|species|subindex} Species ]= Yplus [ Scale_Factor = scale_factor ]`

Parameter	Value	Default
Species	string	undefined
scale_factor	real	undefined

Summary YPLUS model

4.1.649 Pp: Turbulent Kinetic Energy Source

Scope: Aria Material

```
Pp: Turbulent Kinetic Energy Source [ {of | species | subindex} Species ]= Turbulent_Kinetic_Energy_Sou
[ ]
```

Parameter	Value	Default
Species	string	undefined

4.1.650 Pp: Surface Normal Shell

Scope: Aria Material

```
Pp: Surface Normal Shell [ {of | species | subindex} Species ]= Surface_Normal_Shell [ ]
```

Parameter	Value	Default
Species	string	undefined

4.1.651 Pp: Vector Mean Curvature

Scope: Aria Material

```
Pp: Vector Mean Curvature [ {of | species | subindex} Species ]= Vector_Mean_Curvature [ ]
```

Parameter	Value	Default
Species	string	undefined

4.1.652 Prandtl Number

Scope: Aria Material

```
Prandtl Number [ {of | species | subindex} Species ]= Constant [ Pr = Pr ]
```

Parameter	Value	Default
Species	string	undefined
Pr	real	undefined

Summary Constant value

4.1.653 Prandtl Number: Copied

Scope: Aria Material

```
Prandtl Number: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.654 Prandtl Number: User Plugin

Scope: Aria Material

Prandtl Number: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.655 Prandtl Number: General

Scope: Aria Material

Prandtl Number: General [{of | species | subindex} *Species*]= General [ ]

Parameter	Value	Default
Species	string	undefined

Summary General Prandtl number computed from material properties

4.1.656 Prandtl Number: Correlation

Scope: Aria Material

Prandtl Number: Correlation [{of | species | subindex} *Species*]= Correlation [Pr = *Pr* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
Pr	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Prandtl number computed from material properties

4.1.657 Precursor Concentration

Scope: Aria Material

Precursor Concentration [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.658 Precursor Concentration: Copied

Scope: Aria Material

Precursor Concentration: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.659 Precursor Concentration: User Plugin

Scope: Aria Material

Precursor Concentration: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.660 Precursor Conservation

Scope: Aria Material

Precursor Conservation [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.661 Precursor Conservation: Copied

Scope: Aria Material

Precursor Conservation: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.662 Precursor Conservation: User Plugin

Scope: Aria Material

Precursor Conservation: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.663 Pressure

Scope: Aria Material

Pressure [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.664 Pressure: Copied

Scope: Aria Material

Pressure: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.665 Pressure: User Plugin

Scope: Aria Material

Pressure: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parser	"string"	undefined

Summary Value from a user plugin

4.1.666 Pressure: Copy Phase All

Scope: Aria Material

Pressure: Copy Phase All [{of | species | subindex} *Species*]= Copy_Phase_All [ ]

Parameter	Value	Default
Species	string	undefined

Summary Copy the overall multi-phase pressure to the individual phase.

4.1.667 Pressure: From No Material Phase

Scope: Aria Material

Pressure: From No Material Phase [{of | species | subindex} *Species*]= From_No_Material_Phase [ ]

Parameter	Value	Default
Species	string	undefined

Summary Copy the overall pressure to the material phase (i.e. gas phase or solid phase).

4.1.668 Pressure: From Other Material Phase

Scope: Aria Material

Pressure: From Other Material Phase [{of | species | subindex} *Species*]= From_Other_Material_Phase [Other_Phase = *other_phase*]

Parameter	Value	Default
Species	string	undefined
other_phase	"string"	undefined

Summary Copy the overall pressure from a different phase (i.e. gas phase or solid phase).

4.1.669 Pressure: Pressurization Model

Scope: Aria Material

Pressure: Pressurization Model [{of | species | subindex} *Species*]= Pressurization_Model [ ]

Parameter	Value	Default
Species	string	undefined

Summary Pressure from a pressurization model

4.1.670 Pressure: Ideal Gas

Scope: Aria Material

`Pressure: Ideal Gas [ {of|species|subindex} Species ]= Ideal_Gas [ R = r ]`

Parameter	Value	Default
Species	string	undefined
r	real	undefined

Summary Calculate the pressure based on the ideal gas law

4.1.671 Pressure: Equation Of State

Scope: Aria Material

`Pressure: Equation Of State [ {of|species|subindex} Species ]= Equation_Of_State [ Model_Name = model_name ]`

Parameter	Value	Default
Species	string	undefined
model_name	"string"	undefined

Summary Pressure calculated based on species, temperature, and equation of state for the specified pressurization model.

4.1.672 Pressure: From Ideal Gas Density

Scope: Aria Material

`Pressure: From Ideal Gas Density [ {of|species|subindex} Species ]= From_Ideal_Gas_Density [ Mw = mw |R = r ]`

Parameter	Value	Default
Species	string	undefined
mw	real	undefined
r	real	undefined

Summary Calculate pressure using the density and ideal gas law.

4.1.673 Radiation Form Factor

Scope: Aria Material

`Radiation Form Factor [ {of|species|subindex} Species ]= Constant [ F = f ]`

Parameter	Value	Default
Species	string	undefined
f	real	undefined

Summary Constant value

4.1.674 Radiation Form Factor: Copied

Scope: Aria Material

Radiation Form Factor: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.675 Radiation Form Factor: User Plugin

Scope: Aria Material

Radiation Form Factor: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.676 Radiation Form Factor: Calore User Sub

Scope: Aria Material

Radiation Form Factor: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.677 Radiation Form Factor: Fortran

Scope: Aria Material

Radiation Form Factor: Fortran [{of | species | subindex} *Species*]= Fortran [Multiplier = *multiplier* | Sub_Name = *sub_name* | Real_Data = *real_data* | Int_Data = *int_data* | Resource_Name = *resource_name* | Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine emissivity.

4.1.678 Radiative Conductivity

Scope: Aria Material

Radiative Conductivity [{of | species | subindex} *Species*]= Constant [Krad = *Krad*]

Parameter	Value	Default
Species	string	undefined
Krad	real	undefined

Summary Constant value

4.1.679 Radiative Conductivity: Copied

Scope: Aria Material

Radiative Conductivity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.680 Radiative Conductivity: User Plugin

Scope: Aria Material

Radiative Conductivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parameter	"string"	undefined

Summary Value from a user plugin

4.1.681 Radiative Conductivity: Optically Thick

Scope: Aria Material

Radiative Conductivity: Optically Thick [{of | species | subindex} *Species*]= Optically_Thick [Beta_R = *beta_r* | N = *n*]

Parameter	Value	Default
Species	string	undefined
beta_r	real	undefined
n	real	undefined

Summary Radiation conductivity for optically thick materials

4.1.682 Radiative Conductivity: Volume Average

Scope: Aria Material

Radiative Conductivity: Volume Average [{of | species | subindex} *Species*]= Volume_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Radiation conductivity computed from normalized volume average of species radiation conductivities

4.1.683 Radiative Conductivity: Chemeq Foam

Scope: Aria Material

Radiative Conductivity: Chemeq Foam [{of | species | subindex} *Species*]= Chemeq_Foam [Kc0 = *Kc0* | Kc1 = *Kc1*]

Parameter	Value	Default
Species	string	undefined
Kc0	real	undefined
Kc1	real	undefined

Summary Radiation conductivity for ChemEq foams, adjusting for the mass fraction of solids in the foam

4.1.684 Reaction Rate Multiplier

Scope: Aria Material

`Reaction Rate Multiplier [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	`string`	`undefined`
value	`real`	`undefined`

Summary Constant value

4.1.685 Reaction Rate Multiplier: Copied

Scope: Aria Material

`Reaction Rate Multiplier: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.686 Reaction Rate Multiplier: User Plugin

Scope: Aria Material

`Reaction Rate Multiplier: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.687 Reaction Rate Multiplier: One Minus Gas Volume Fraction

Scope: Aria Material

`Reaction Rate Multiplier: One Minus Gas Volume Fraction [ {of | species | subindex} Species ]= One_Minus_Gas_Volume_Fraction [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Compute a multiplier that is one minus the gas volume fractions

4.1.688 Reaction Rate Multiplier: Vitrification

Scope: Aria Material

`Reaction Rate Multiplier: Vitrification [ {of|species|subindex} `*`Species`*` ]= Vitrification [ W = `*`w`*` |Beta = `*`beta`*` ]`

Parameter	Value	Default
Species	string	undefined
w	real	undefined
beta	real	undefined

Summary Compute vitrification rate multiplier from polymerization shift factor

4.1.689 Relative Permeability

Scope: Aria Material

`Relative Permeability [ {of|species|subindex} `*`Species`*` ]= Constant [ K = `*`k`*` ]`

Parameter	Value	Default
Species	string	undefined
k	real	undefined

Summary Constant value

4.1.690 Relative Permeability: Copied

Scope: Aria Material

`Relative Permeability: Copied [ {of|species|subindex} `*`Species`*` ]= Copied [ Source = `*`source`*` ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.691 Relative Permeability: User Plugin

Scope: Aria Material

`Relative Permeability: User Plugin [ {of|species|subindex} `*`Species`*` ]= User_Plugin [ Name = `*`Name`*` |`*`magic_trailing_string_vector_param`*` ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.692 Relative Permeability: T Exponent

Scope: Aria Material

Relative Permeability: T Exponent [{of | species | subindex} *Species*]= T_Exponent [K_Ref = *k_ref* | T_Ref = *t_ref* | N = *n*]

Parameter	Value	Default
Species	string	undefined
k_ref	real	undefined
t_ref	real	undefined
n	real	undefined

Summary Relative permeability as a power of normalized temperature

4.1.693 Relative Permeability: Volume Average

Scope: Aria Material

Relative Permeability: Volume Average [{of | species | subindex} *Species*]= Volume_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Relative permeability computed from volume average of species relative permeabilities

4.1.694 Relative Permeability: Mass Average

Scope: Aria Material

Relative Permeability: Mass Average [{of | species | subindex} *Species*]= Mass_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Linear mixture-averaged mass diffusivity

4.1.695 Relative Permeability: Van Genuchten Gas

Scope: Aria Material

Relative Permeability: Van Genuchten Gas [{of | species | subindex} *Species*]= Van_Genuchten_Gas [Liquid_Phase_Name = *liquid_phase_name* | Beta_Field = *beta_field* | Sgr_Field = *sgr_field* | Slr_Field = *slr_field* | Beta = *beta* | Residual_Liquid_Phase_Saturation = *residual_liquid_phase_saturatio* | Residual_Gas_Phase_Saturation = *residual_gas_phase_saturation*]

Parameter	Value	Default
Species	string	undefined
liquid_phase_name	"string"	undefined
beta_field	"string"	undefined
sgr_field	"string"	undefined
slr_field	"string"	undefined
beta	real	undefined
residual_liquid_phase_saturation	real	undefined
residual_gas_phase_saturation	real	undefined

Summary A Van Genuchten gas model for relative permeability

4.1.696 Relative Permeability: Udell Cubic Gas

Scope: Aria Material

Relative Permeability: Udell Cubic Gas [{of|species|subindex} *Species*]= Udell_Cubic_Gas
[Liquid_Phase_Name = *liquid_phase_name* |Residual_Liquid_Phase_Saturation = *residual_liquid_phase_satu*
|Residual_Gas_Phase_Saturation = *residual_gas_phase_saturation*]

Parameter	Value	Default
Species	string	undefined
liquid_phase_name	"string"	undefined
residual_liquid_phase_saturation	real	undefined
residual_gas_phase_saturation	real	undefined

Summary A Udell cubic gas model for relative permeability

4.1.697 Relative Permeability: Van Genuchten Liquid

Scope: Aria Material

Relative Permeability: Van Genuchten Liquid [{of|species|subindex} *Species*]= Van_Genuchten_Liqui
[Liquid_Phase_Name = *liquid_phase_name* |Beta_Field = *beta_field* |Sgr_Field = *sgr_field* |
Slr_Field = *slr_field* |Beta = *beta* |Residual_Liquid_Phase_Saturation = *residual_liquid_phase_saturatio*
|Residual_Gas_Phase_Saturation = *residual_gas_phase_saturation*]

Parameter	Value	Default
Species	string	undefined
liquid_phase_name	"string"	undefined
beta_field	"string"	undefined
sgr_field	"string"	undefined
slr_field	"string"	undefined
beta	real	undefined
residual_liquid_phase_saturation	real	undefined
residual_gas_phase_saturation	real	undefined

Summary A Van Genuchten liquid model for relative permeability

4.1.698 Relative Permeability: Udell Cubic Liquid

Scope: Aria Material

Relative Permeability: Udell Cubic Liquid [{of | species | subindex} *Species*]= Udell_Cubic_Liquid [Liquid_Phase_Name = *liquid_phase_name* | Residual_Liquid_Phase_Saturation = *residual_liquid_phase_satu* | Residual_Gas_Phase_Saturation = *residual_gas_phase_saturation*]

Parameter	Value	Default
Species	string	undefined
liquid_phase_name	"string"	undefined
residual_liquid_phase_saturation	real	undefined
residual_gas_phase_saturation	real	undefined

Summary A Udell cubic liquid model for relative permeability

4.1.699 Relative Permeability: Brooks Corey

Scope: Aria Material

Relative Permeability: Brooks Corey [{of | species | subindex} *Species*]= Brooks_Corey [Liquid_Phase_Name = *liquid_phase_name*]

Parameter	Value	Default
Species	string	undefined
liquid_phase_name	"string"	undefined

Summary A Brooks-Corey liquid/gas model for relative permeability

4.1.700 Reservoir Depth

Scope: Aria Material

Reservoir Depth [{of | species | subindex} *Species*]= Constant [H = *h*]

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.701 Reservoir Depth: Copied

Scope: Aria Material

Reservoir Depth: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.702 Reservoir Depth: User Plugin

Scope: Aria Material

Reservoir Depth: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.703 Residual Saturation

Scope: Aria Material

Residual Saturation [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.704 Residual Saturation: Copied

Scope: Aria Material

Residual Saturation: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.705 Residual Saturation: User Plugin

Scope: Aria Material

Residual Saturation: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.706 Retardation: Saturated Distributed

Scope: Aria Material

`Retardation: Saturated Distributed [ {of | species | subindex} Species ]= Saturated_Distributed [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Distributed retardation factor for saturated porous flow. The corresponding distribution coefficient should be defined with a DISTRIBUTION COEFFICIENT model.

4.1.707 Retardation: Gas Phase Distributed

Scope: Aria Material

`Retardation: Gas Phase Distributed [ {of | species | subindex} Species ]= Gas_Phase_Distributed [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Distributed retardation factor for gas phase in porous flow. The corresponding distribution coefficient should be defined with a DISTRIBUTION COEFFICIENT model.

4.1.708 Retardation: Liquid Phase Distributed

Scope: Aria Material

`Retardation: Liquid Phase Distributed [ {of | species | subindex} Species ]= Liquid_Phase_Distributed [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Distributed retardation factor for liquid phase porous flow. The corresponding distribution coefficient should be defined with a DISTRIBUTION COEFFICIENT model.

4.1.709 Retardation: Wetting Phase Distributed

Scope: Aria Material

`Retardation: Wetting Phase Distributed [ {of | species | subindex} Species ]= Wetting_Phase_Distributed [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Distributed retardation factor for wetting phase porous flow. The corresponding distribution coefficient should be defined with a DISTRIBUTION COEFFICIENT model.

4.1.710 Retardation: Nonwetting Phase Distributed

Scope: Aria Material

`Retardation: Nonwetting Phase Distributed [ {of | species | subindex} Species ]= Nonwetting_Phase_Distri`
`[ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Distributed retardation factor for non-wetting phase in porous flow. The corresponding distribution coefficient should be defined with a DISTRIBUTION COEFFICIENT model.

4.1.711 Retardation

Scope: Aria Material

`Retardation [ {of | species | subindex} Species ]= Constant [ R = r ]`

Parameter	Value	Default
Species	`string`	`undefined`
r	`real`	`undefined`

Summary Constant value

4.1.712 Retardation: Copied

Scope: Aria Material

`Retardation: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.713 Retardation: User Plugin

Scope: Aria Material

`Retardation: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name`
`| magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.714 Reynolds Number

Scope: Aria Material

`Reynolds Number [ {of | species | subindex} Species ]= Constant [ Re = Re ]`

Parameter	Value	Default
Species	`string`	`undefined`
Re	`real`	`undefined`

Summary Constant value

4.1.715 Reynolds Number: Copied

Scope: Aria Material

`Reynolds Number: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.716 Reynolds Number: User Plugin

Scope: Aria Material

`Reynolds Number: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.717 Reynolds Number: General

Scope: Aria Material

`Reynolds Number: General [ {of | species | subindex} Species ]= General [ L = l ]`

Parameter	Value	Default
Species	`string`	`undefined`
l	`real`	`undefined`

Summary Reynold number computed from a specified length scale

4.1.718 Reynolds Number: Correlation

Scope: Aria Material

Reynolds Number: Correlation [{of | species | subindex} *Species*]= Correlation [Re = *Re* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
Re	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Reynolds number computed from material properties

4.1.719 Saturated Retardation

Scope: Aria Material

Saturated Retardation [{of | species | subindex} *Species*]= Constant [R = *r*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined

Summary Constant value

4.1.720 Saturated Retardation: Copied

Scope: Aria Material

Saturated Retardation: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.721 Saturated Retardation: User Plugin

Scope: Aria Material

Saturated Retardation: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.722 Saturation

Scope: Aria Material

Saturation [{of | species | subindex} *Species*]= Constant [S = *s*]

Parameter	Value	Default
Species	string	undefined
s	real	undefined

Summary Constant value

4.1.723 Saturation: Copied

Scope: Aria Material

Saturation: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.724 Saturation: User Plugin

Scope: Aria Material

Saturation: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.725 Saturation: Van Genuchten

Scope: Aria Material

Saturation: Van Genuchten [{of | species | subindex} *Species*]= Van_Genuchten [Is_Wetting_Phase = *is_wetting_phase* | Nonwetting_Phase = *nonwetting_phase* | Reference_Capillary_Pressure = *reference_capi* | Beta = *beta* | Residual_Wetting_Phase_Saturation = *residual_wetting_phase_saturation* | Residual_Nonwetti = *residual_nonwetting_phase_saturation* | Wetting_Phase = *wetting_phase*]

Parameter	Value	Default
Species	string	undefined
is_wetting_phase	integer	undefined
nonwetting_phase	"string"	undefined
reference_capillary_pressure	real	undefined
beta	real	undefined
residual_wetting_phase_saturation	real	undefined
residual_nonwetting_phase_saturation	real	undefined
wetting_phase	"string"	undefined

Summary A Van Genuchten model for saturation using capillary pressure

4.1.726 Saturation: From Other Phase

Scope: Aria Material

```
Saturation: From Other Phase [ {of | species | subindex} Species ]= From_Other_Phase [ ]
```

Parameter	Value	Default
Species	string	undefined

Summary Get the saturation from one minus the other phase saturation

4.1.727 Scaling

Scope: Aria Material

```
Scaling [ {of | species | subindex} Species ]= Constant [ Value = value ]
```

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.728 Scaling: Copied

Scope: Aria Material

```
Scaling: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.729 Scaling: User Plugin

Scope: Aria Material

Scaling: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_*
]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.730 Scattering Coefficient

Scope: Aria Material

Scattering Coefficient [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.731 Scattering Coefficient: Copied

Scope: Aria Material

Scattering Coefficient: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*
]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.732 Scattering Coefficient: User Plugin

Scope: Aria Material

Scattering Coefficient: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name
= *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.733 Scattering Cross Section

Scope: Aria Material

Scattering Cross Section [{of | species | subindex} *Species*]= Constant [Scat = *scat*]

Parameter	Value	Default
Species	string	undefined
scat	real	undefined

Summary Constant value

4.1.734 Scattering Cross Section: Copied

Scope: Aria Material

Scattering Cross Section: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.735 Scattering Cross Section: User Plugin

Scope: Aria Material

Scattering Cross Section: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.736 Scattering Cross Section: Calore User Sub

Scope: Aria Material

Scattering Cross Section: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Name = *name* | Type = *type* | Multiplier = *multiplier* | Material_Data_Block = *material_data_block* | Data = *data* | Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Calore user subroutine scattering cross section

4.1.737 Schmidt Number

Scope: Aria Material

Schmidt Number [{of | species | subindex} *Species*]= Constant [Sc = *Sc*]

Parameter	Value	Default
Species	string	undefined
Sc	real	undefined

Summary Constant value

4.1.738 Schmidt Number: Copied

Scope: Aria Material

Schmidt Number: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.739 Schmidt Number: User Plugin

Scope: Aria Material

Schmidt Number: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.740 Seebeck Coefficient

Scope: Aria Material

`Seebeck Coefficient [ {of|species|subindex} Species ]= Constant [ A = a ]`

Parameter	Value	Default
Species	string	undefined
a	real	undefined

Summary Constant value

4.1.741 Seebeck Coefficient: Copied

Scope: Aria Material

`Seebeck Coefficient: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.742 Seebeck Coefficient: User Plugin

Scope: Aria Material

`Seebeck Coefficient: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.743 Shell Lofting Factor

Scope: Aria Material

`Shell Lofting Factor [ {of|species|subindex} Species ]= Constant [ F = f ]`

Parameter	Value	Default
Species	string	undefined
f	real	undefined

Summary Constant value

4.1.744 Shell Lofting Factor: Copied

Scope: Aria Material

Shell Lofting Factor: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.745 Shell Lofting Factor: User Plugin

Scope: Aria Material

Shell Lofting Factor: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.746 Shell Lofting Factor: Element Attribute

Scope: Aria Material

Shell Lofting Factor: Element Attribute [{of | species | subindex} *Species*]= Element_Attribute [Multiplier = *multiplier* | Attribute_Name = *attribute_name*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
attribute_name	"string"	undefined

Summary Element Distribution Factor for the lofting of shell surface

4.1.747 Shell Thickness

Scope: Aria Material

Shell Thickness [{of | species | subindex} *Species*]= Constant [T = *t*]

Parameter	Value	Default
Species	string	undefined
t	real	undefined

Summary Constant value

4.1.748 Shell Thickness: Copied

Scope: Aria Material

Shell Thickness: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.749 Shell Thickness: User Plugin

Scope: Aria Material

Shell Thickness: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.750 Shell Thickness: Element Attribute

Scope: Aria Material

Shell Thickness: Element Attribute [{of|species|subindex} *Species*]= Element_Attribute [Multiplier = *multiplier* |Attribute_Name = *attribute_name* |Contact = *contact*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
attribute_name	"string"	undefined
contact	"string"	undefined

Summary Distribution Factor for the shell thickness

4.1.751 Skeleton Density

Scope: Aria Material

Skeleton Density [{of|species|subindex} *Species*]= Constant [Rho = *rho*]

Parameter	Value	Default
Species	string	undefined
rho	real	undefined

Summary Constant value

4.1.752 Skeleton Density: Copied

Scope: Aria Material

`Skeleton Density: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.753 Skeleton Density: User Plugin

Scope: Aria Material

`Skeleton Density: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.754 Skeleton Enthalpy: Cpt

Scope: Aria Material

`Skeleton Enthalpy: Cpt [ {of | species | subindex} Species ]= Cpt [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary CPT model for the the solid skeleton enthalpy

4.1.755 Skeleton Internal Energy

Scope: Aria Material

`Skeleton Internal Energy [ {of | species | subindex} Species ]= Constant [ E = e ]`

Parameter	Value	Default
Species	`string`	`undefined`
e	`real`	`undefined`

Summary Constant value

4.1.756 Skeleton Internal Energy: Copied

Scope: Aria Material

Skeleton Internal Energy: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.757 Skeleton Internal Energy: User Plugin

Scope: Aria Material

Skeleton Internal Energy: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.758 Skeleton Internal Energy: Cpt

Scope: Aria Material

Skeleton Internal Energy: Cpt [{of | species | subindex} *Species*]= Cpt [ ]

Parameter	Value	Default
Species	string	undefined

Summary CPT model for the the solid skeleton internal energy

4.1.759 Skeleton Internal Energy: Linear

Scope: Aria Material

Skeleton Internal Energy: Linear [{of | species | subindex} *Species*]= Linear [Cp = *cp* | T_Ref = *T_ref*]

Parameter	Value	Default
Species	string	undefined
cp	real	undefined
T_ref	real	undefined

Summary CPT model for the the solid skeleton internal energy

4.1.760 Skeleton Specific Heat

Scope: Aria Material

`Skeleton Specific Heat [ {of | species | subindex} `*`Species`*` ]= Constant [ Cp = `*`cp`*` ]`

Parameter	Value	Default
Species	string	undefined
cp	real	undefined

Summary Constant value

4.1.761 Skeleton Specific Heat: Copied

Scope: Aria Material

`Skeleton Specific Heat: Copied [ {of | species | subindex} `*`Species`*` ]= Copied [ Source = `*`source`*` ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.762 Skeleton Specific Heat: User Plugin

Scope: Aria Material

`Skeleton Specific Heat: User Plugin [ {of | species | subindex} `*`Species`*` ]= User_Plugin [ Name = `*`Name`*` | `*`magic_trailing_string_vector_param`*` ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.763 Solid Body Acceleration

Scope: Aria Material

`Solid Body Acceleration [ {of | species | subindex} `*`Species`*` ]= Constant [ Valuex = `*`valuex`*` | Valuey = `*`valuey`*` | Valuez = `*`valuez`*` ]`

Parameter	Value	Default
Species	string	undefined
valuex	real	undefined
valuey	real	undefined
valuez	real	undefined

Summary Constant value

4.1.764 Solid Body Acceleration: User Plugin

Scope: Aria Material

Solid Body Acceleration: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.765 Solid Density

Scope: Aria Material

Solid Density [{of | species | subindex} *Species*]= Constant [Rho_Solid = *rho_solid*]

Parameter	Value	Default
Species	string	undefined
rho_solid	real	undefined

Summary Constant value

4.1.766 Solid Density: Copied

Scope: Aria Material

Solid Density: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.767 Solid Density: User Plugin

Scope: Aria Material

Solid Density: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.768 Solid Lambda

Scope: Aria Material

`Solid Lambda [ {of | species | subindex} `*`Species`*` ]= Constant [ Lambda = `*`lambda`*` ]`

Parameter	Value	Default
Species	string	undefined
lambda	real	undefined

Summary Constant value

4.1.769 Solid Lambda: Copied

Scope: Aria Material

`Solid Lambda: Copied [ {of | species | subindex} `*`Species`*` ]= Copied [ Source = `*`source`*` ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.770 Solid Lambda: User Plugin

Scope: Aria Material

`Solid Lambda: User Plugin [ {of | species | subindex} `*`Species`*` ]= User_Plugin [ Name = `*`Name`*` | `*`magic_trailing_string_vector_param`*` ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.771 Solid Lambda: Converted

Scope: Aria Material

`Solid Lambda: Converted [ {of | species | subindex} `*`Species`*` ]= Converted [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Lame coefficient computed from Poisson's ratio and elastic modulus

4.1.772 Solid Lambda: Converted Plane Stress

Scope: Aria Material

`Solid Lambda: Converted Plane Stress [ {of|species|subindex} Species ]= Converted_Plane_Stress [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Lame coefficient applicable to plane stress conditions computed from Poisson's ratio and elastic modulus

4.1.773 Solid Poissons Ratio

Scope: Aria Material

`Solid Poissons Ratio [ {of|species|subindex} Species ]= Constant [ Nu = nu ]`

Parameter	Value	Default
Species	string	undefined
nu	real	undefined

Summary Constant value

4.1.774 Solid Poissons Ratio: Copied

Scope: Aria Material

`Solid Poissons Ratio: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.775 Solid Poissons Ratio: User Plugin

Scope: Aria Material

`Solid Poissons Ratio: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.776 Solid Pressure Gradient: Gidaspow

Scope: Aria Material

Solid Pressure Gradient: Gidaspow [{of|species|subindex} *Species*]= Gidaspow [G0 = *g0* |C = *c* |Fmax = *fmax*]

Parameter	Value	Default
Species	string	undefined
g0	real	undefined
c	real	undefined
fmax	real	undefined

4.1.777 Solid Stress: Isothermal

Scope: Aria Material

Solid Stress: Isothermal [{of|species|subindex} *Species*]= Isothermal [T = *t* |T_Ref = *t_ref*]

Parameter	Value	Default
Species	string	undefined
t	real	undefined
t_ref	real	undefined

4.1.778 Solid Stress: Linear Elastic

Scope: Aria Material

Solid Stress: Linear Elastic [{of|species|subindex} *Species*]= Linear_Elastic [Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.779 Solid Stress: Porous Effective

Scope: Aria Material

Solid Stress: Porous Effective [{of|species|subindex} *Species*]= Porous_Effective [Pressure = *pressure* |Phase = *phase* |Biot = *biot*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined

Summary Apply the pressure as an isotropic mesh stress. Primarily used as an effective stress in porous flow problems.

4.1.780 Solid Stress: Saturation Weighted Porous Effective

Scope: Aria Material

Solid Stress: Saturation Weighted Porous Effective [{of | species | subindex} *Species*]= Saturation_Weighted_Porous_Effective [Pressure = *pressure* | Phase = *phase* | Biot = *biot*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined

Summary Apply the pressure as an isotropic mesh stress. Primarily used as an effective stress in porous flow problems.

4.1.781 Solid Stress: Mooney Rivlin

Scope: Aria Material

Solid Stress: Mooney Rivlin [{of | species | subindex} *Species*]= Mooney_Rivlin [ ]

Parameter	Value	Default
Species	string	undefined

4.1.782 Solid Stress: Neohookean Elastic

Scope: Aria Material

Solid Stress: Neohookean Elastic [{of | species | subindex} *Species*]= Neohookean_Elastic [ ]

Parameter	Value	Default
Species	string	undefined

4.1.783 Solid Stress: Incompressible Newtonian

Scope: Aria Material

Solid Stress: Incompressible Newtonian [{of | species | subindex} *Species*]= Incompressible_Newtonian [Pressure = *pressure* | Phase = *phase* | Biot = *biot* | Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
pressure	"string"	undefined
phase	"string"	undefined
biot	real	undefined
reference_frame	"string"	undefined

4.1.784 Solid Stress: Nonlinear Elastic

Scope: Aria Material

Solid Stress: Nonlinear Elastic [{of|species|subindex} *Species*]= Nonlinear_Elastic [ ]

Parameter	Value	Default
Species	string	undefined

4.1.785 Solid Stress: Residual

Scope: Aria Material

Solid Stress: Residual [{of|species|subindex} *Species*]= Residual [Sz = *sz* |Szz = *szz* |Sxz = *sxz* |Syz = *syz* |Sy = *sy* |Syy = *syy* |Sxy = *sxy* |Sx = *sx* |Sxx = *sxx*]

Parameter	Value	Default
Species	string	undefined
sz	real	undefined
szz	real	undefined
sxz	real	undefined
syz	real	undefined
sy	real	undefined
syy	real	undefined
sxy	real	undefined
sx	real	undefined
sxx	real	undefined

4.1.786 Solid Stress: Thermal

Scope: Aria Material

Solid Stress: Thermal [{of|species|subindex} *Species*]= Thermal [ ]

Parameter	Value	Default
Species	string	undefined

4.1.787 Solid Stress: Species

Scope: Aria Material

Solid Stress: Species [{of|species|subindex} *Species*]= Species [Sref = *sref* |Phase = *phase*]

Parameter	Value	Default
Species	string	undefined
sref	real	undefined
phase	"string"	undefined

Summary A linear mechanical strain due to a species concentration

4.1.788 Solid Stress: Species Transversely Isotropic

Scope: Aria Material

Solid Stress: Species Transversely Isotropic [{of|species|subindex} *Species*]= Species_Transversel
[Refval = *refval* |Beta_N = *beta_n* |Beta_T = *beta_t* |N_X = *n_x* |N_Y = *n_y* |N_Z = *n_z* |
Phase = *phase*]

Parameter	Value	Default
Species	string	undefined
refval	real	undefined
beta_n	real	undefined
beta_t	real	undefined
n_x	real	undefined
n_y	real	undefined
n_z	real	undefined
phase	"string"	undefined

Summary A linear mechanical strain that is transversely isotropic due to a species concentration

4.1.789 Solid Stress: Species Anisotropic

Scope: Aria Material

Solid Stress: Species Anisotropic [{of|species|subindex} *Species*]= Species_Anisotropic
[Sref = *sref* |Phase = *phase*]

Parameter	Value	Default
Species	string	undefined
sref	real	undefined
phase	"string"	undefined

Summary Anisotropic linear mechanical strain due to a species concentration

4.1.790 Solid Stress: Lame

Scope: Aria Material

Solid Stress: Lame [{of|species|subindex} *Species*]= Lame [Model = *model*]

Parameter	Value	Default
Species	string	undefined
model	"string"	undefined

4.1.791 Solid Two Mu

Scope: Aria Material

Solid Two Mu [{of|species|subindex} *Species*]= Constant [Two_Mu = *two_mu*]

Parameter	Value	Default
Species	string	undefined
two_mu	real	undefined

Summary Constant value

4.1.792 Solid Two Mu: Copied

Scope: Aria Material

`Solid Two Mu: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.793 Solid Two Mu: User Plugin

Scope: Aria Material

`Solid Two Mu: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.794 Solid Two Mu: Converted

Scope: Aria Material

`Solid Two Mu: Converted [ {of | species | subindex} Species ]= Converted [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Lame coefficient computed from Poisson's ratio and elastic modulus

4.1.795 Solid Youngs Modulus

Scope: Aria Material

`Solid Youngs Modulus [ {of | species | subindex} Species ]= Constant [ E = e ]`

Parameter	Value	Default
Species	string	undefined
e	real	undefined

Summary Constant value

4.1.796 Solid Youngs Modulus: Copied

Scope: Aria Material

`Solid Youngs Modulus: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.797 Solid Youngs Modulus: User Plugin

Scope: Aria Material

`Solid Youngs Modulus: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.798 Soret Coefficient

Scope: Aria Material

`Soret Coefficient [ {of|species|subindex} Species ]= Constant [ S = s ]`

Parameter	Value	Default
Species	string	undefined
s	real	undefined

Summary Constant value

4.1.799 Soret Coefficient: Copied

Scope: Aria Material

`Soret Coefficient: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.800 Soret Coefficient: User Plugin

Scope: Aria Material

Soret Coefficient: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.801 Sound Speed

Scope: Aria Material

Sound Speed [{of | species | subindex} *Species*]= Constant [C = *c*]

Parameter	Value	Default
Species	string	undefined
c	real	undefined

4.1.802 Species

Scope: Aria Material

Species [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.803 Species: Copied

Scope: Aria Material

Species: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.804 Species: User Plugin

Scope: Aria Material

Species: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_*
]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parameter	"string"	undefined

Summary Value from a user plugin

4.1.805 Species: Partial Molar Volume

Scope: Aria Material

Species: Partial Molar Volume [{of|species|subindex} *Species*]= Partial_Molar_Volume
[ ]

Parameter	Value	Default
Species	string	undefined

4.1.806 Species: Sum All Species

Scope: Aria Material

Species: Sum All Species [{of|species|subindex} *Species*]= Sum_All_Species [ ]

Parameter	Value	Default
Species	string	undefined

4.1.807 Species: Solvent Concentration From Solute Concentrations And Partial Molar Volumes

Scope: Aria Material

Species: Solvent Concentration From Solute Concentrations And Partial Molar Volumes [{of
|species|subindex} *Species*]= Solvent_Concentration_From_Solute_Concentrations_And_Partial_Molar_Volume
[ ]

Parameter	Value	Default
Species	string	undefined

4.1.808 Species: Chemeq Gas

Scope: Aria Material

Species: Chemeq Gas [{of|species|subindex} *Species*]= Chemeq_Gas [ ]

Parameter	Value	Default
Species	string	undefined

4.1.809 Species: From Chemeq

Scope: Aria Material

`Species: From Chemeq [ {of | species | subindex} Species ]= From_Chemeq [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Species concentration of one species extracted from ChemEq

4.1.810 Species: From Mass Fraction

Scope: Aria Material

`Species: From Mass Fraction [ {of | species | subindex} Species ]= From_Mass_Fraction [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Molar concentration of a single species calculated from its mass fraction, molecular weight, and overall density

4.1.811 Species: Phase Average

Scope: Aria Material

`Species: Phase Average [ {of | species | subindex} Species ]= Phase_Average [ Subindex_A = subindex_a |Subindex_B = subindex_b ]`

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.812 Species: From Phase All

Scope: Aria Material

`Species: From Phase All [ {of | species | subindex} Species ]= From_Phase_All [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Copy the species from PHASE_ALL to the desired phase.

4.1.813 Species: From Material Phase

Scope: Aria Material

Species: From Material Phase [{of|species|subindex} *Species*]= From_Material_Phase [Phase_Name = *phase_name*]

Parameter	Value	Default
Species	string	undefined
phase_name	"string"	undefined

Summary Copy the species from another material phase to the desired material phase.

4.1.814 Species: From Density

Scope: Aria Material

Species: From Density [{of|species|subindex} *Species*]= From_Density [ ]

Parameter	Value	Default
Species	string	undefined

4.1.815 Species Diffusion: Ficks Law

Scope: Aria Material

Species Diffusion: Ficks Law [{of|species|subindex} *Species*]= Ficks_Law [ ]

Parameter	Value	Default
Species	string	undefined

4.1.816 Species Diffusion: Nernst Planck

Scope: Aria Material

Species Diffusion: Nernst Planck [{of|species|subindex} *Species*]= Nernst_Planck [F = *f* |R = *r*]

Parameter	Value	Default
Species	string	undefined
f	real	undefined
r	real	undefined

4.1.817 Species Diffusion: All Subindices Use Stefan Maxwell

Scope: Aria Material

Species Diffusion: All Subindices Use Stefan Maxwell [{of|species|subindex} *Species*]= All_Subindices_Use_Stefan_Maxwell [Velocityconversion = *velocityConversion* |Canteraxmlfile = *canteraXMLFile* |Velocitybasis = *velocityBasis*]

Parameter	Value	Default
Species	string	undefined
velocityConversion	real	undefined
canteraXMLFile	"string"	undefined
velocityBasis	"string"	undefined

4.1.818 Species Diffusion: Basic

Scope: Aria Material

Species Diffusion: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

4.1.819 Species Diffusion: Constant Tensorial

Scope: Aria Material

Species Diffusion: Constant Tensorial [{of | species | subindex} *Species*]= Constant_Tensorial [D_Xx = *d_xx* | D_Xy = *d_xy* | D_Xz = *d_xz* | D_Yx = *d_yx* | D_Yy = *d_yy* | D_Yz = *d_yz* | D_Zx = *d_zx* | D_Zy = *d_zy* | D_Zz = *d_zz*]

Parameter	Value	Default
Species	string	undefined
d_xx	real	undefined
d_xy	real	undefined
d_xz	real	undefined
d_yx	real	undefined
d_yy	real	undefined
d_yz	real	undefined
d_zx	real	undefined
d_zy	real	undefined
d_zz	real	undefined

4.1.820 Species Diffusion: Tensorial

Scope: Aria Material

Species Diffusion: Tensorial [{of | species | subindex} *Species*]= Tensorial [ ]

Parameter	Value	Default
Species	string	undefined

4.1.821 Species Diffusion: Tensorial Dispersive

Scope: Aria Material

Species Diffusion: Tensorial Dispersive [{of | species | subindex} *Species*]= Tensorial_Dispersive [Alpha_T = *alpha_t* | Alpha_L = *alpha_l* | Tau = *tau*]

Parameter	Value	Default
Species	string	undefined
alpha_t	real	undefined
alpha_l	real	undefined
tau	real	undefined

4.1.822 Species Diffusion: Thermophoresis

Scope: Aria Material

`Species Diffusion: Thermophoresis [ {of | species | subindex} Species ]= Thermophoresis [ ]`

Parameter	Value	Default
Species	string	undefined

4.1.823 Species Diffusion: Tensor Thermophoresis

Scope: Aria Material

`Species Diffusion: Tensor Thermophoresis [ {of | species | subindex} Species ]= Tensor_Thermophoresis [ ]`

Parameter	Value	Default
Species	string	undefined

4.1.824 Species Diffusion: Electromigration

Scope: Aria Material

`Species Diffusion: Electromigration [ {of | species | subindex} Species ]= Electromigration [ ]`

Parameter	Value	Default
Species	string	undefined

4.1.825 Species Diffusion: Tensor Electromigration

Scope: Aria Material

`Species Diffusion: Tensor Electromigration [ {of | species | subindex} Species ]= Tensor_Electromigrat [ ]`

Parameter	Value	Default
Species	string	undefined

4.1.826 Species Diffusion: Darcy

Scope: Aria Material

`Species Diffusion: Darcy [ {of | species | subindex} Species ]= Darcy [ ]`

Parameter	Value	Default
Species	string	undefined

4.1.827 Species Diffusion: Chemical Potential

Scope: Aria Material

Species Diffusion: Chemical Potential [{of | species | subindex} *Species*]= Chemical_Potential [ ]

Parameter	Value	Default
Species	string	undefined

Summary Model species diffusion using a flux determined from the gradient of the chemical potential. $\vec{J} = -Mc\nabla\mu$ where M is the mobility, c the species concentration, and mu the chemical potential.

4.1.828 Species Diffusion: Tensor Chemical Potential

Scope: Aria Material

Species Diffusion: Tensor Chemical Potential [{of | species | subindex} *Species*]= Tensor_Chemical_Po [ ]

Parameter	Value	Default
Species	string	undefined

Summary Model species diffusion using a flux determined from the gradient of the chemical potential. $\vec{J} = -Mc\nabla\mu$ where M is the tensor mobility, c the species concentration, and mu the chemical potential.

4.1.829 Species Diffusion: Mass Balance Fracbal

Scope: Aria Material

Species Diffusion: Mass Balance Fracbal [{of | species | subindex} *Species*]= Mass_Balance_Fracbal [ ]

Parameter	Value	Default
Species	string	undefined

Summary Species diffusion for the fracbal species in a mass balance equation system. $\vec{J}_i = -1 \sum_{j \neq i} (m\vec{dot}_j)/MW_i$.

4.1.830 Species Diffusivity

Scope: Aria Material

Species Diffusivity [{of | species | subindex} *Species*]= Constant [D = *d*]

Parameter	Value	Default
Species	string	undefined
d	real	undefined

Summary Constant valueConstant value

4.1.831 Species Diffusivity: Copied

Scope: Aria Material

Species Diffusivity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expressionCopied value from another expression

4.1.832 Species Diffusivity: User Plugin

Scope: Aria Material

Species Diffusivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user pluginValue from a user plugin

4.1.833 Species Diffusivity: Arrhenius

Scope: Aria Material

Species Diffusivity: Arrhenius [{of | species | subindex} *Species*]= Arrhenius [R = *r* | C = *c* | D = *d* | Q = *q*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
c	real	undefined
d	real	undefined
q	real	undefined

Summary Arrhenius species diffusivityArrhenius species diffusivity

4.1.834 Species Diffusivity: Dissolution

Scope: Aria Material

Species Diffusivity: Dissolution [{of | species | subindex} *Species*]= Dissolution [D = *d*]

Parameter	Value	Default
Species	string	undefined
d	"string"	undefined

Summary Dissolution species diffusivity for a species disolving in a liquid from a gas.Dissolution species diffusivity for a species disolving in a liquid from a gas.

4.1.835 Species Expansion Coeff

Scope: Aria Material

Species Expansion Coeff [{of | species | subindex} *Species*]= Constant [Species_Expansion_Coeff = *species_expansion_coeff*]

Parameter	Value	Default
Species	string	undefined
species_expansion_coeff	real	undefined

Summary Constant value

4.1.836 Species Expansion Coeff: Copied

Scope: Aria Material

Species Expansion Coeff: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.837 Species Expansion Coeff: User Plugin

Scope: Aria Material

Species Expansion Coeff: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.838 Species Face Stabilization Scaling: Default

Scope: Aria Material

Summary Default face stabilization scaling for energy equation

4.1.839 Species Fraction

Scope: Aria Material

`Species Fraction [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.840 Species Fraction: Copied

Scope: Aria Material

`Species Fraction: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.841 Species Fraction: User Plugin

Scope: Aria Material

`Species Fraction: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.842 Species Fraction: From Chemeq

Scope: Aria Material

`Species Fraction: From Chemeq [ {of | species | subindex} Species ]= From_Chemeq [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Species fraction of one species exracted from ChemEq

4.1.843 Species Fraction: From Species

Scope: Aria Material

`Species Fraction: From Species [ {of | species | subindex} Species ]= From_Species [ ]`

Parameter	Value	Default
Species	`string`	`undefined`

Summary Species fraction of one species from it's concentration and the overall concentration

4.1.844 Species Mobility

Scope: Aria Material

`Species Mobility [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	`string`	`undefined`
value	`real`	`undefined`

Summary Constant value

4.1.845 Species Mobility: Copied

Scope: Aria Material

`Species Mobility: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.846 Species Mobility: User Plugin

Scope: Aria Material

`Species Mobility: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.847 Species Mobility: Nernst Einstein

Scope: Aria Material

`Species Mobility: Nernst Einstein [ {of|species|subindex} Species ]= Nernst_Einstein [ R = r |Temperature_Material_Phase = temperature_material_phase ]`

Parameter	Value	Default
Species	string	undefined
r	real	undefined
temperature_material_phase	"string"	undefined

Summary Mobility from diffusivity, M = D/(RT).

4.1.848 Species Production: From Time Rate Of Change

Scope: Aria Material

`Species Production: From Time Rate Of Change [ {of|species|subindex} Species ]= From_Time_Rate_Of_C [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Species production rate from the time derivative of species concentration.

4.1.849 Species Surface

Scope: Aria Material

`Species Surface [ {of|species|subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.850 Species Surface: Copied

Scope: Aria Material

`Species Surface: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.851 Species Surface: User Plugin

Scope: Aria Material

Species Surface: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.852 Species Surface: Equivalent To Bulk

Scope: Aria Material

Species Surface: Equivalent To Bulk [{of | species | subindex} *Species*]= Equivalent_To_Bulk [ ]

Parameter	Value	Default
Species	string	undefined

Summary Approximate a surface species concentration as identical to the bulk concentration

4.1.853 Species Surface: Diffusion To Surf Correction

Scope: Aria Material

Species Surface: Diffusion To Surf Correction [{of | species | subindex} *Species*]= Diffusion_To_Surf [Species_A = *species_a* | Species_B = *species_b* | D = *d* | R = *r* | F = *f* | K = *k* | Eqpot = *eqPot* | Dir = *dir*]

Parameter	Value	Default
Species	string	undefined
species_a	"string"	undefined
species_b	"string"	undefined
d	real	undefined
r	real	undefined
f	real	undefined
k	real	undefined
eqPot	real	undefined
dir	real	undefined

Summary Correction for surface concentration based on balancing reaction rate and diffusion from pore center to wall.

4.1.854 Species Valence

Scope: Aria Material

Species Valence [{of | species | subindex} *Species*]= Constant [Z = *z*]

Parameter	Value	Default
Species	string	undefined
z	real	undefined

Summary Constant value

4.1.855 Species Valence: Copied

Scope: Aria Material

Species Valence: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.856 Species Valence: User Plugin

Scope: Aria Material

Species Valence: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.857 Specific Dissipation Rate

Scope: Aria Material

Specific Dissipation Rate [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.858 Specific Dissipation Rate: Copied

Scope: Aria Material

Specific Dissipation Rate: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.859 Specific Dissipation Rate: User Plugin

Scope: Aria Material

Specific Dissipation Rate: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.860 Specific Dissipation Rate Density

Scope: Aria Material

Specific Dissipation Rate Density [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.861 Specific Dissipation Rate Density: Copied

Scope: Aria Material

Specific Dissipation Rate Density: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.862 Specific Dissipation Rate Density: User Plugin

Scope: Aria Material

Specific Dissipation Rate Density: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.863 Specific Dissipation Rate Diffusive Flux: Basic

Scope: Aria Material

Specific Dissipation Rate Diffusive Flux: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Basic model of specific dissipation rate diffusive flux

4.1.864 Specific Heat: Calore User Sub

Scope: Aria Material

Specific Heat: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Name = *name* | Type = *type* | Multiplier = *multiplier* | Material_Data_Block = *material_data_block* | Data = *data* | Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.865 Specific Heat: Curing Foam

Scope: Aria Material

Specific Heat: Curing Foam [{of | species | subindex} *Species*]= Curing_Foam [ ]

Parameter	Value	Default
Species	string	undefined

Summary Specific heat for a curing foam

4.1.866 Specific Heat: Phase Average

Scope: Aria Material

Specific Heat: Phase Average [{of | species | subindex} *Species*]= Phase_Average [Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.867 Specific Heat: Use Phase Change

Scope: Aria Material

Specific Heat: Use Phase Change [{of | species | subindex} *Species*]= Use_Phase_Change [Flh = *flh* |Ts = *ts* |Tl = *tl* |Type = *type* |Field = *field*]

Parameter	Value	Default
Species	string	undefined
flh	real	undefined
ts	real	undefined
tl	real	undefined
type	"string"	undefined
field	"string"	undefined

Summary Specific heat with a step function accounting for phase change.

Known Issue: Using this model can cause poor convergence. Use the 'Melting' source for energy for much better numerical performance.

4.1.868 Specific Heat: Interpolated Phase Average

Scope: Aria Material

Specific Heat: Interpolated Phase Average [{of | species | subindex} *Species*]= Interpolated_Phase_Ave [Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.869 Specific Heat: Porous Phase Specific Average

Scope: Aria Material

Specific Heat: Porous Phase Specific Average [{of | species | subindex} *Species*]= Porous_Phase_Speci
[Liquid_Phase_Name = *Liquid_Phase_Name* |Gas_Phase_Name = *Gas_Phase_Name*]

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined

Summary Specific heat for a multiphase material of two or three phases

4.1.870 Specific Heat: Mass Average

Scope: Aria Material

Specific Heat: Mass Average [{of | species | subindex} *Species*]= Mass_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Specific heat computed from mass average of species specific heats

4.1.871 Specific Heat: Cantera

Scope: Aria Material

Specific Heat: Cantera [{of | species | subindex} *Species*]= Cantera [Poffset = *poffset*
]

Parameter	Value	Default
Species	string	undefined
poffset	real	undefined

Summary Cantera specific heat for low Mach number applications

4.1.872 Specific Heat

Scope: Aria Material

Specific Heat [{of | species | subindex} *Species*]= Constant [H0 = *h0* |S0 = *s0* |T_Standard
= *T_standard* |Cp = *Cp*]

Parameter	Value	Default
Species	string	undefined
h0	real	undefined
s0	real	undefined
T_standard	real	undefined
Cp	real	undefined

Summary Constant specific heat from an evaluator

4.1.873 Specific Heat: T Exponent

Scope: Aria Material

Specific Heat: T Exponent [{of|species|subindex} *Species*]= T_Exponent [H0 = *h0* |S0 = *s0* |T_Standard = *T_standard* |Cp_Ref = *Cp_ref* |T_Ref = *T_ref* |N = *n*]

Parameter	Value	Default
Species	string	undefined
h0	real	undefined
s0	real	undefined
T_standard	real	undefined
Cp_ref	real	undefined
T_ref	real	undefined
n	real	undefined

Summary Temperature exponent specific heat from an evaluator

4.1.874 Specific Heat: Nasa14

Scope: Aria Material

Specific Heat: Nasa14 [{of|species|subindex} *Species*]= Nasa14 [H0 = *h0* |S0 = *s0* |T_Standard = *T_standard* |Tmid = *Tmid* |Tmax = *Tmax* |R = *r* |L0 = *l0* |L1 = *l1* |L2 = *l2* |L3 = *l3* |L4 = *l4* |L5 = *l5* |L6 = *l6* |H1 = *h1* |H2 = *h2* |H3 = *h3* |H4 = *h4* |H5 = *h5* |H6 = *h6* |T_Offset = *t_offset*]

Parameter	Value	Default
Species	string	undefined
h0	real	undefined
s0	real	undefined
T_standard	real	undefined
Tmid	real	undefined
Tmax	real	undefined
r	real	undefined
l0	real	undefined
l1	real	undefined
l2	real	undefined
l3	real	undefined
l4	real	undefined
l5	real	undefined
l6	real	undefined
h1	real	undefined
h2	real	undefined
h3	real	undefined
h4	real	undefined
h5	real	undefined
h6	real	undefined
t_offset	real	undefined

Summary Value from a NASA14 polynomial function

4.1.875 Specific Heat: Fortran

Scope: Aria Material

Specific Heat: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine specific heat.

4.1.876 Specific Heat Cp

Scope: Aria Material

Specific Heat Cp [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.877 Specific Heat Cp: Copied

Scope: Aria Material

`Specific Heat Cp: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.878 Specific Heat Cp: User Plugin

Scope: Aria Material

`Specific Heat Cp: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_param	`"string"`	`undefined`

Summary Value from a user plugin

4.1.879 Specific Heat Cp: Cantera

Scope: Aria Material

`Specific Heat Cp: Cantera [ {of | species | subindex} Species ]= Cantera [ Poffset = poffset ]`

Parameter	Value	Default
Species	`string`	`undefined`
poffset	`real`	`undefined`

Summary Cantera specific heat for low Mach number applications

4.1.880 Specific Surface Area

Scope: Aria Material

`Specific Surface Area [ {of | species | subindex} Species ]= Constant [ Value = value ]`

Parameter	Value	Default
Species	`string`	`undefined`
value	`real`	`undefined`

Summary Constant value

4.1.881 Specific Surface Area: Copied

Scope: Aria Material

Specific Surface Area: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.882 Specific Surface Area: User Plugin

Scope: Aria Material

Specific Surface Area: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.883 Supg Tau: Shakib Energy

Scope: Aria Material

Supg Tau: Shakib Energy = Shakib_Energy [H = *h* | Use_Advection = *use_advection* | Use_Diffusion = *use_diffusion* | Use_Time = *use_time* | Multiplier = *multiplier*]

Parameter	Value	Default
h	real	undefined
use_advection	integer	undefined
use_diffusion	integer	undefined
use_time	integer	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.884 Supg Tau: Shakib Enthalpy

Scope: Aria Material

Supg Tau: Shakib Enthalpy = Shakib_Enthalpy [Multiplier = *multiplier* | Use_Diffusion = *use_diffusion* | Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.885 Supg Tau: Shakib Momentum

Scope: Aria Material

Supg Tau: Shakib Momentum = Shakib_Momentum [H = *h* |Use_Advection = *use_advection* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time* |Multiplier = *multiplier*]

Parameter	Value	Default
h	real	undefined
use_advection	integer	undefined
use_diffusion	integer	undefined
use_time	integer	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.886 Supg Tau: Shakib Level Set

Scope: Aria Material

Supg Tau: Shakib Level Set = Shakib_Level_Set [H = *h* |Use_Advection = *use_advection* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time* |Multiplier = *multiplier*]

Parameter	Value	Default
h	real	undefined
use_advection	integer	undefined
use_diffusion	integer	undefined
use_time	integer	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.887 Supg Tau: Shakib Suspension

Scope: Aria Material

Supg Tau: Shakib Suspension = Shakib_Suspension [H = *h* |Use_Advection = *use_advection* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time* |Multiplier = *multiplier*]

Parameter	Value	Default
h	real	undefined
use_advection	integer	undefined
use_diffusion	integer	undefined
use_time	integer	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model for the suspension equation

4.1.888 Supg Tau: Classic Energy

Scope: Aria Material

Supg Tau: Classic Energy = Classic_Energy [Epsilon = *epsilon* |Multiplier = *multiplier*]

Parameter	Value	Default
epsilon	real	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.889 Supg Tau: Classic Momentum

Scope: Aria Material

Supg Tau: Classic Momentum = Classic_Momentum [Epsilon = *epsilon* |Multiplier = *multiplier*]

Parameter	Value	Default
epsilon	real	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.890 Supg Tau: Shakib Cvfem Momentum

Scope: Aria Material

Supg Tau: Shakib Cvfem Momentum = Shakib_Cvfem_Momentum [Multiplier = *multiplier* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.891 Supg Tau: Shakib Cvfem Mixture Fraction

Scope: Aria Material

Supg Tau: Shakib Cvfem Mixture Fraction = Shakib_Cvfem_Mixture_Fraction [Multiplier = *multiplier* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.892 Supg Tau: Shakib Cvfem Level Set

Scope: Aria Material

Supg Tau: Shakib Cvfem Level Set = Shakib_Cvfem_Level_Set [Multiplier = *multiplier* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.893 Supg Tau: Shakib Cvfem Dispersed Phase Momentum

Scope: Aria Material

Supg Tau: Shakib Cvfem Dispersed Phase Momentum = Shakib_Cvfem_Dispersed_Phase_Momentum [Multiplier = *multiplier* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.894 Supg Tau: Classic Mixture Fraction

Scope: Aria Material

Supg Tau: Classic Mixture Fraction = Classic_Mixture_Fraction [Epsilon = *epsilon* |Multiplier = *multiplier*]

Parameter	Value	Default
epsilon	real	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.895 Supg Tau: Shakib Mixture Fraction

Scope: Aria Material

Supg Tau: Shakib Mixture Fraction = Shakib_Mixture_Fraction [Multiplier = *multiplier* | Use_Diffusion = *use_diffusion* |Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.896 Supg Tau: Classic Species

Scope: Aria Material

Supg Tau: Classic Species = Classic_Species [Epsilon = *epsilon* |Multiplier = *multiplier*]

Parameter	Value	Default
epsilon	real	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.897 Supg Tau: Shakib Species

Scope: Aria Material

Supg Tau: Shakib Species = Shakib_Species [Multiplier = *multiplier* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.898 Supg Tau: Shakib Mass Balance

Scope: Aria Material

Supg Tau: Shakib Mass Balance = Shakib_Mass_Balance [Multiplier = *multiplier* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time*]

Parameter	Value	Default
multiplier	real	undefined
use_diffusion	integer	undefined
use_time	integer	undefined

Summary A Shakib SUPG Tau model

4.1.899 Supg Tau: Shakib Charge Density

Scope: Aria Material

Supg Tau: Shakib Charge Density = Shakib_Charge_Density [H = *h* |Use_Advection = *use_advection* |Use_Diffusion = *use_diffusion* |Use_Time = *use_time* |Multiplier = *multiplier*]

Parameter	Value	Default
h	real	undefined
use_advection	integer	undefined
use_diffusion	integer	undefined
use_time	integer	undefined
multiplier	real	undefined

Summary A Shakib SUPG Tau model

4.1.900 Surface Tension

Scope: Aria Material

Surface Tension [{of|species|subindex} *Species*]= Constant [Sigma = *sigma*]

Parameter	Value	Default
Species	string	undefined
sigma	real	undefined

Summary Constant value

4.1.901 Surface Tension: Copied

Scope: Aria Material

Surface Tension: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.902 Surface Tension: User Plugin

Scope: Aria Material

Surface Tension: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.903 Surface Tension: Linear T

Scope: Aria Material

Surface Tension: Linear T [{of | species | subindex} *Species*]= Linear_T [Sigma0 = *sigma0* | Dsigmadt = *dsigmadT* | T_Ref = *T_ref*]

Parameter	Value	Default
Species	string	undefined
sigma0	real	undefined
dsigmadT	real	undefined
T_ref	real	undefined

Summary Linear T surface tension

4.1.904 Suspension Flux: Phillips

Scope: Aria Material

Suspension Flux: Phillips [{of | species | subindex} *Species*]= Phillips [K_Mu = *k_mu* | K_C = *k_c* | Phi_Max = *phi_max* | Beta = *beta* | Particle_Radius = *particle_radius* | K_0 = *k_0* | Phi_Tol = *phi_tol* | Use_Fd = *use_fd*]

Parameter	Value	Default
Species	string	undefined
k_mu	real	undefined
k_c	real	undefined
phi_max	real	undefined
beta	real	undefined
particle_radius	real	undefined
k_0	real	undefined
phi_tol	real	undefined
use_fd	integer	undefined

4.1.905 Suspension Flux: Fad Phillips

Scope: Aria Material

Suspension Flux: Fad Phillips [{of | species | subindex} *Species*]= Fad_Phillips [K_Mu = *k_mu* | K_C = *k_c* | Phi_Max = *phi_max* | Beta = *beta* | Particle_Radius = *particle_radius* | K_0 = *k_0* | Phi_Tol = *phi_tol*]

Parameter	Value	Default
Species	string	undefined
k_mu	real	undefined
k_c	real	undefined
phi_max	real	undefined
beta	real	undefined
particle_radius	real	undefined
k_0	real	undefined
phi_tol	real	undefined

4.1.906 Suspension Flux: Balance

Scope: Aria Material

Suspension Flux: Balance [{of|species|subindex} *Species*]= Balance [A = *a* |Mu_S = *mu_s* |Lambda_1 = *lambda_1* |Lambda_2 = *lambda_2* |Lambda_3 = *lambda_3*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined
mu_s	real	undefined
lambda_1	real	undefined
lambda_2	real	undefined
lambda_3	real	undefined

4.1.907 Suspension Flux: Hydrostatic

Scope: Aria Material

Suspension Flux: Hydrostatic [{of|species|subindex} *Species*]= Hydrostatic [Gx = *gx* |Gy = *gy* |Gz = *gz* |Lambda = *lambda* |Fluid_Density = *fluid_density* |B = *b* |Phi_Max = *phi_max*]

Parameter	Value	Default
Species	string	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined
lambda	real	undefined
fluid_density	real	undefined
b	real	undefined
phi_max	real	undefined

Summary Hydrostatic source for suspension equation

4.1.908 Suspension Hindrance Function: Morris

Scope: Aria Material

Suspension Hindrance Function: Morris [{of|species|subindex} *Species*]= Morris [Alpha = *alpha* |Phi_Max = *phi_max* |Phi_Tol = *phi_tol*]

Parameter	Value	Default
Species	string	undefined
alpha	real	undefined
phi_max	real	undefined
phi_tol	real	undefined

4.1.909 Suspension Normal Viscosity: Morris

Scope: Aria Material

Suspension Normal Viscosity: Morris [{of|species|subindex} *Species*]= Morris [K_N = *K_n* |Phi_Max = *phi_max* |Phi_Tol = *phi_tol* |Mu_S = *mu_s*]

Parameter	Value	Default
Species	string	undefined
K_n	real	undefined
phi_max	real	undefined
phi_tol	real	undefined
mu_s	real	undefined

4.1.910 Suspension Q Tensor: Isotropic

Scope: Aria Material

Suspension Q Tensor: Isotropic [{of|species|subindex} *Species*]= Isotropic [ ]

Parameter	Value	Default
Species	string	undefined

4.1.911 Suspension Q Tensor: Ct Aligned

Scope: Aria Material

Suspension Q Tensor: Ct Aligned [{of|species|subindex} *Species*]= Ct_Aligned [Lambda_1 = *lambda_1* |Lambda_2 = *lambda_2* |Lambda_3 = *lambda_3*]

Parameter	Value	Default
Species	string	undefined
lambda_1	real	undefined
lambda_2	real	undefined
lambda_3	real	undefined

4.1.912 Suspension Q Tensor: Flow Aligned

Scope: Aria Material

Suspension Q Tensor: Flow Aligned [{of|species|subindex} *Species*]= Flow_Aligned_ [Lambda_A = *lambda_a* |Lambda_B = *lambda_b*]

Parameter	Value	Default
Species	string	undefined
lambda_a	real	undefined
lambda_b	real	undefined

4.1.913 Species Names

Scope: Aria Material

Species Names {=|are|is} *Names...*

Parameter	Value	Default
Names	string...	undefined

Description This defines the available chemical species, if they are not already specified by an external property TPL.

4.1.914 Temperature

Scope: Aria Material

Temperature [{of|species|subindex} *Species*]= Constant [Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Value = *value*]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
value	real	undefined

Summary Constant value

4.1.915 Temperature: Copied

Scope: Aria Material

Temperature: Copied [{of|species|subindex} *Species*]= Copied [Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Source = *source*]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.916 Temperature: User Plugin

Scope: Aria Material

Temperature: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Name = *Name* | *magic_trailing_string_vecto*
]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
Name	"string"	undefined
magic_trailing_string_vector_parser	"string"	undefined

Summary Value from a user plugin

4.1.917 Temperature: Calore User Sub

Scope: Aria Material

Temperature: Calore User Sub [{of | species | subindex} *Species*]= Calore_User_Sub [Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Multiplier = *multiplier* | Name = *name* | Type = *type* | Material_Data_Block = *material_data_block* | Data = *data*]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined
name	"string"	undefined
type	"string"	undefined
material_data_block	"string"	undefined
data	"string"	undefined

Summary Values from a Calore user subroutine

4.1.918 Temperature: From No Material Phase

Scope: Aria Material

Temperature: From No Material Phase [{of | species | subindex} *Species*]= From_No_Material_Phase [Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle*]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined

Summary Copy the temperature from NO_MATERIAL_PHASE to the desired material phase.

4.1.919 Temperature: Cantera

Scope: Aria Material

Temperature: Cantera [{of|species|subindex} *Species*]= Cantera [Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Omit_Fd_Sens = *omit_fd_sens*]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
omit_fd_sens	integer	undefined

Summary Cantera enthalpy computed from temperature

4.1.920 Temperature: Fortran User Sub

Scope: Aria Material

Temperature: Fortran User Sub [{of|species|subindex} *Species*]= Fortran_User_Sub [Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Multiplier = *multiplier* |Name = *name* |Type = *type* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data* |Material_Data_Block = *material_data_block*]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined
name	"string"	undefined
type	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined
material_data_block	"string"	undefined

Summary Dirichlet value from a Fortran user subroutine

4.1.921 Temperature: Cht Robin

Scope: Aria Material

Temperature: Cht Robin [{of|species|subindex} *Species*]= Cht_Robin [Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Temperature_Field = *temperature_field*]

Parameter	Value	Default
Species	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
temperature_field	"string"	undefined

Summary Dirichlet boundary condition for interface coupled with Robin style BC CHT_ROBIN. This BC computes the heat flux for the other side.

4.1.922 Temperature: Correlation Wall

Scope: Aria Material

Temperature: Correlation Wall [{of|species|subindex} *Species*]= Correlation_Wall [Tw = *Tw* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
Tw	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation wall temperature

4.1.923 Temperature: Correlation Fluid

Scope: Aria Material

Temperature: Correlation Fluid [{of|species|subindex} *Species*]= Correlation_Fluid [T = *t* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
t	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation fluid temperature

4.1.924 Tensor Bulk Conductivity

Scope: Aria Material

Tensor Bulk Conductivity [{of|species|subindex} *Species*]= Constant [Xx = *xx* |T11 = *t11* |Xy = *xy* |T12 = *t12* |Xz = *xz* |T13 = *t13* |Yx = *yx* |T21 = *t21* |Yy = *yy* |T22 = *t22* |Yz = *yz* |T23 = *t23* |Zx = *zx* |T31 = *t31* |Zy = *zy* |T32 = *t32* |Zz = *zz* |T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the bulk conductivity.

4.1.925 Tensor Bulk Conductivity: Diagonal

Scope: Aria Material

`Tensor Bulk Conductivity: Diagonal [ {of | species | subindex} Species ]= Diagonal [ Variable = variable ]`

Parameter	Value	Default
Species	string	undefined
variable	"string"	undefined

4.1.926 Tensor Bulk Conductivity: Volume Average

Scope: Aria Material

`Tensor Bulk Conductivity: Volume Average [ {of | species | subindex} Species ]= Volume_Average [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Bulk conductivity tensor computed from volume average of species bulk conductivities

4.1.927 Tensor Bulk Conductivity Scaling

Scope: Aria Material

`Tensor Bulk Conductivity Scaling [ {of | species | subindex} Species ]= Constant [ S = s ]`

Parameter	Value	Default
Species	string	undefined
s	real	undefined

Summary Constant value

4.1.928 Tensor Bulk Conductivity Scaling: Copied

Scope: Aria Material

```
Tensor Bulk Conductivity Scaling: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.929 Tensor Bulk Conductivity Scaling: User Plugin

Scope: Aria Material

```
Tensor Bulk Conductivity Scaling: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.930 Tensor Bulk Conductivity Scaling: T Exponent

Scope: Aria Material

```
Tensor Bulk Conductivity Scaling: T Exponent [ {of|species|subindex} Species ]= T_Exponent [ Kbulk_Ref = kbulk_ref |T_Ref = t_ref |N = n ]
```

Parameter	Value	Default
Species	string	undefined
kbulk_ref	real	undefined
t_ref	real	undefined
n	real	undefined

Summary Tensor thermal conductivity scaling for bulk material. Nominally this returns $k = k_{ref}\left(\frac{T}{T_{ref}}\right)^n$. If you provide expressions for BULK_DENSITY and SOLID_DENSITY this is also scaled by the ratio of SOLID_DENSITY/BULK_DENSITY

4.1.931 Tensor Electrical Conductivity: Calore User Sub

Scope: Aria Material

Tensor Electrical Conductivity: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined

Summary Tensor values for temperature dependent electrical conductivity from a calore user subroutine

4.1.932 Tensor Electrical Conductivity

Scope: Aria Material

Tensor Electrical Conductivity [{of|species|subindex} *Species*]= Constant [Xx = *xx* |T11 = *t11* |Xy = *xy* |T12 = *t12* |Xz = *xz* |T13 = *t13* |Yx = *yx* |T21 = *t21* |Yy = *yy* |T22 = *t22* |Yz = *yz* |T23 = *t23* |Zx = *zx* |T31 = *t31* |Zy = *zy* |T32 = *t32* |Zz = *zz* |T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the electrical conductivity

4.1.933 Tensor Electromigration Coefficient

Scope: Aria Material

Tensor Electromigration Coefficient [{of|species|subindex} *Species*]= Constant [Xx = *xx* |T11 = *t11* |Xy = *xy* |T12 = *t12* |Xz = *xz* |T13 = *t13* |Yx = *yx* |T21 = *t21* |Yy = *yy* |T22 = *t22* |Yz = *yz* |T23 = *t23* |Zx = *zx* |T31 = *t31* |Zy = *zy* |T32 = *t32* |Zz = *zz* |T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the Soret coefficient

4.1.934 Tensor Soret Coefficient

Scope: Aria Material

Tensor Soret Coefficient [{of|species|subindex} *Species*]= Constant [Xx = *xx* |T11 = *t11* |Xy = *xy* |T12 = *t12* |Xz = *xz* |T13 = *t13* |Yx = *yx* |T21 = *t21* |Yy = *yy* |T22 = *t22* |Yz = *yz* |T23 = *t23* |Zx = *zx* |T31 = *t31* |Zy = *zy* |T32 = *t32* |Zz = *zz* |T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the Soret coefficient

4.1.935 Tensor Species Diffusivity

Scope: Aria Material

Tensor Species Diffusivity [{of | species | subindex} *Species*]= Constant [Xx = *xx* | T11 = *t11* | Xy = *xy* | T12 = *t12* | Xz = *xz* | T13 = *t13* | Yx = *yx* | T21 = *t21* | Yy = *yy* | T22 = *t22* | Yz = *yz* | T23 = *t23* | Zx = *zx* | T31 = *t31* | Zy = *zy* | T32 = *t32* | Zz = *zz* | T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the species diffusivity

4.1.936 Tensor Species Expansion Coeff

Scope: Aria Material

Tensor Species Expansion Coeff [{of | species | subindex} *Species*]= Constant [Xx = *xx* | T11 = *t11* | Xy = *xy* | T12 = *t12* | Xz = *xz* | T13 = *t13* | Yx = *yx* | T21 = *t21* | Yy = *yy* | T22 = *t22* | Yz = *yz* | T23 = *t23* | Zx = *zx* | T31 = *t31* | Zy = *zy* | T32 = *t32* | Zz = *zz* | T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the stress coefficient due to species concentration.

4.1.937 Tensor Species Mobility: Nernst Einstein

Scope: Aria Material

Tensor Species Mobility: Nernst Einstein [{of | species | subindex} *Species*]= Nernst_Einstein [R = *r* | Temperature_Material_Phase = *temperature_material_phase*]

Parameter	Value	Default
Species	string	undefined
r	real	undefined
temperature_material_phase	"string"	undefined

Summary Tensor mobility from diffusivity, M = D/(RT).

4.1.938 Tensor Species Mobility

Scope: Aria Material

Tensor Species Mobility [{of | species | subindex} *Species*]= Constant [Xx = *xx* | T11 = *t11* | Xy = *xy* | T12 = *t12* | Xz = *xz* | T13 = *t13* | Yx = *yx* | T21 = *t21* | Yy = *yy* | T22 = *t22* | Yz = *yz* | T23 = *t23* | Zx = *zx* | T31 = *t31* | Zy = *zy* | T32 = *t32* | Zz = *zz* | T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the electrical conductivity

4.1.939 Tensor Thermal Conductivity: Calore User Sub

Scope: Aria Material

Tensor Thermal Conductivity: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined

Summary Tensor values for the thermal conductivity from a calore user subroutine

4.1.940 Tensor Thermal Conductivity

Scope: Aria Material

Tensor Thermal Conductivity [{of|species|subindex} *Species*]= Constant [Xx = *xx* |T11 = *t11* |Xy = *xy* |T12 = *t12* |Xz = *xz* |T13 = *t13* |Yx = *yx* |T21 = *t21* |Yy = *yy* |T22 = *t22* |Yz = *yz* |T23 = *t23* |Zx = *zx* |T31 = *t31* |Zy = *zy* |T32 = *t32* |Zz = *zz* |T33 = *t33*]

Parameter	Value	Default
Species	string	undefined
xx	real	undefined
t11	real	undefined
xy	real	undefined
t12	real	undefined
xz	real	undefined
t13	real	undefined
yx	real	undefined
t21	real	undefined
yy	real	undefined
t22	real	undefined
yz	real	undefined
t23	real	undefined
zx	real	undefined
t31	real	undefined
zy	real	undefined
t32	real	undefined
zz	real	undefined
t33	real	undefined

Summary Constant tensor value for the thermal conductivity

4.1.941 Tensor Thermal Conductivity: Mesh Input

Scope: Aria Material

Tensor Thermal Conductivity: Mesh Input [{of|species|subindex} *Species*]= Mesh_Input [Multiplier = *multiplier* |Name = *name* |Type = *type* |Axis1 = *axis1* |Axis2 = *axis2* |Tolerance = *tolerance*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
name	"string"	undefined
type	"string"	undefined
axis1	"string"	undefined
axis2	"string"	undefined
tolerance	real	undefined

Summary Tensor thermal conductivity from the input mesh database

4.1.942 Tensor Thermal Conductivity: Mass Average

Scope: Aria Material

Tensor Thermal Conductivity: Mass Average [{of|species|subindex} *Species*]= Mass_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Thermal conductivity computed from mass average of species conductivities

4.1.943 Tensor Thermal Conductivity: Summed

Scope: Aria Material

Tensor Thermal Conductivity: Summed [{of | species | subindex} *Species*]= Summed [Contributions = *contributions*]

Parameter	Value	Default
Species	string	undefined
contributions	"string"	undefined

Summary Tensor thermal conductivity as a sum of user-specified conductivities

4.1.944 Tensor Thermal Conductivity: Level Set Aligned

Scope: Aria Material

Tensor Thermal Conductivity: Level Set Aligned [{of | species | subindex} *Species*]= Level_Set_Aligned [K_Normal = *k_normal* | K_Tangent = *k_tangent* | Level_Set_Subindex = *level_set_subindex*]

Parameter	Value	Default
Species	string	undefined
k_normal	real	undefined
k_tangent	real	undefined
level_set_subindex	integer	undefined

Summary Tensor values for the thermal conductivity from a calore user subroutine

4.1.945 Thermal Conductivity

Scope: Aria Material

Thermal Conductivity [{of | species | subindex} *Species*]= Constant [K = *k*]

Parameter	Value	Default
Species	string	undefined
k	real	undefined

Summary Constant value

4.1.946 Thermal Conductivity: Copied

Scope: Aria Material

Thermal Conductivity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.947 Thermal Conductivity: User Plugin

Scope: Aria Material

Thermal Conductivity: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.948 Thermal Conductivity: Calore User Sub

Scope: Aria Material

Thermal Conductivity: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

4.1.949 Thermal Conductivity: Cantera

Scope: Aria Material

Thermal Conductivity: Cantera [{of|species|subindex} *Species*]= Cantera [Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
poffset	real	undefined

Summary Cantera thermal conductivity

4.1.950 Thermal Conductivity: Curing Foam

Scope: Aria Material

Thermal Conductivity: Curing Foam [{of|species|subindex} *Species*]= Curing_Foam [ ]

Parameter	Value	Default
Species	string	undefined

4.1.951 Thermal Conductivity: From Prandtl

Scope: Aria Material

`Thermal Conductivity: From Prandtl [ {of|species|subindex} Species ]= From_Prandtl [ ]`

Parameter	Value	Default
Species	string	undefined

Summary ?

4.1.952 Thermal Conductivity: Optically Thick

Scope: Aria Material

`Thermal Conductivity: Optically Thick [ {of|species|subindex} Species ]= Optically_Thick [ K = k |Beta_R = beta_r |N = n ]`

Parameter	Value	Default
Species	string	undefined
k	real	undefined
beta_r	real	undefined
n	real	undefined

Summary Thermal conductivity model for optically thick materials

4.1.953 Thermal Conductivity: Phase Average

Scope: Aria Material

`Thermal Conductivity: Phase Average [ {of|species|subindex} Species ]= Phase_Average [ Subindex_A = subindex_a |Subindex_B = subindex_b ]`

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.954 Thermal Conductivity: Interpolated Phase Average

Scope: Aria Material

`Thermal Conductivity: Interpolated Phase Average [ {of|species|subindex} Species ]= Interpolated_Pl [ Subindex_A = subindex_a |Subindex_B = subindex_b ]`

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.955 Thermal Conductivity: Power Law

Scope: Aria Material

Thermal Conductivity: Power Law [{of|species|subindex} *Species*]= Power_Law [A = *a* | Gamma = *gamma*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined
gamma	real	undefined

Summary Thermal conductivity as a power law in temperature

4.1.956 Thermal Conductivity: Saturation Power Law

Scope: Aria Material

Thermal Conductivity: Saturation Power Law [{of|species|subindex} *Species*]= Saturation_Power_Law [K_Dry = *k_dry* |K_Wet = *k_wet* |Power = *power*]

Parameter	Value	Default
Species	string	undefined
k_dry	real	undefined
k_wet	real	undefined
power	real	undefined

Summary Thermal conductivity as a power law in liquid saturation for two phase porous flows

4.1.957 Thermal Conductivity: Thermal

Scope: Aria Material

Thermal Conductivity: Thermal [{of|species|subindex} *Species*]= Thermal [A = *a* |B = *b* |C = *c* |D = *d*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined
b	real	undefined
c	real	undefined
d	real	undefined

Summary Thermal conductivity as a cubit polynomial in temperature

4.1.958 Thermal Conductivity: T Exponent

Scope: Aria Material

`Thermal Conductivity: T Exponent [ {of|species|subindex} Species ]= T_Exponent [ K_Ref = k_ref |T_Ref = t_ref |N = n ]`

Parameter	Value	Default
Species	string	undefined
k_ref	real	undefined
t_ref	real	undefined
n	real	undefined

Summary Thermal conductivity as a power of normalized temperature

4.1.959 Thermal Conductivity: Volume Average

Scope: Aria Material

`Thermal Conductivity: Volume Average [ {of|species|subindex} Species ]= Volume_Average [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Thermal conductivity computed from volume average of species thermal conductivities

4.1.960 Thermal Conductivity: Summed

Scope: Aria Material

`Thermal Conductivity: Summed [ {of|species|subindex} Species ]= Summed [ Contributions = contributions ]`

Parameter	Value	Default
Species	string	undefined
contributions	"string"	undefined

Summary Thermal conductivity as a sum of user-specified conductivities

4.1.961 Thermal Conductivity: Activation User Function

Scope: Aria Material

`Thermal Conductivity: Activation User Function [ {of|species|subindex} Species ]= Activation_User_F [ Background = background |Name = name ]`

Parameter	Value	Default
Species	string	undefined
background	real	undefined
name	"string"	undefined

Summary Temperature user function thermal conductivity model activated by an activation Field.

4.1.962 Thermal Conductivity: Linear Temperature And Density

Scope: Aria Material

Thermal Conductivity: Linear Temperature And Density [{of|species|subindex} *Species*]= Linear_Temperature_And_Density [C_0 = *c_0* |C_Rho = *C_rho* |C_T = *c_t*]

Parameter	Value	Default
Species	string	undefined
c_0	real	undefined
C_rho	real	undefined
c_t	real	undefined

Summary Thermal conductivity that is a linear function of both temperature and density. $\kappa = C_0 + C_\rho \rho + C_T T$

4.1.963 Thermal Conductivity: Mass Average

Scope: Aria Material

Thermal Conductivity: Mass Average [{of|species|subindex} *Species*]= Mass_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Thermal conductivity computed from mass average of species conductivities

4.1.964 Thermal Conductivity: Porous Arithmetic Mixture

Scope: Aria Material

Thermal Conductivity: Porous Arithmetic Mixture [{of|species|subindex} *Species*]= Porous_Arithmeti [Liquid_Phase_Name = *Liquid_Phase_Name* |Gas_Phase_Name = *Gas_Phase_Name* |Initial_Saturation = *Initial_Saturation* |Melt_Dt = *melt_DT*]

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined
Initial_Saturation	real	undefined
melt_DT	real	undefined

Summary Thermal conductivity computed arithmetic mean of phase properties

4.1.965 Thermal Conductivity: Porous Harmonic Mixture

Scope: Aria Material

Thermal Conductivity: Porous Harmonic Mixture [{of|species|subindex} *Species*]= Porous_Harmonic_Mi [Liquid_Phase_Name = *Liquid_Phase_Name* |Gas_Phase_Name = *Gas_Phase_Name* |Initial_Saturation = *Initial_Saturation* |Melt_Dt = *melt_DT*]

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined
Initial_Saturation	real	undefined
melt_DT	real	undefined

Summary Thermal conductivity computed harmonic mean of phase properties

4.1.966 Thermal Conductivity: Porous Geometric Mixture

Scope: Aria Material

Thermal Conductivity: Porous Geometric Mixture [{of|species|subindex} *Species*]= Porous_Geometric
[Liquid_Phase_Name = *Liquid_Phase_Name* |Gas_Phase_Name = *Gas_Phase_Name* |Initial_Saturation
= *Initial_Saturation* |Melt_Dt = *melt_DT*]

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined
Initial_Saturation	real	undefined
melt_DT	real	undefined

Summary Thermal conductivity computed geometric mean of phase properties

4.1.967 Thermal Conductivity: Porous Emt Mixture

Scope: Aria Material

Thermal Conductivity: Porous Emt Mixture [{of|species|subindex} *Species*]= Porous_Emt_Mixture
[Liquid_Phase_Name = *Liquid_Phase_Name* |Gas_Phase_Name = *Gas_Phase_Name* |Initial_Saturation
= *Initial_Saturation* |Melt_Dt = *melt_DT*]

Parameter	Value	Default
Species	string	undefined
Liquid_Phase_Name	"string"	undefined
Gas_Phase_Name	"string"	undefined
Initial_Saturation	real	undefined
melt_DT	real	undefined

Summary Thermal conductivity computed effective medium theory model

4.1.968 Thermal Conductivity: Fortran

Scope: Aria Material

Thermal Conductivity: Fortran [{of|species|subindex} *Species*]= Fortran [Multiplier
= *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name
= *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine thermal conductivity.

4.1.969 Thermal Diffusivity

Scope: Aria Material

```
Thermal Diffusivity [ {of | species | subindex} Species ]= Constant [ D = d ]
```

Parameter	Value	Default
Species	string	undefined
d	real	undefined

Summary Constant value

4.1.970 Thermal Diffusivity: Copied

Scope: Aria Material

```
Thermal Diffusivity: Copied [ {of | species | subindex} Species ]= Copied [ Source = source
]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.971 Thermal Diffusivity: User Plugin

Scope: Aria Material

```
Thermal Diffusivity: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name
= Name | magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.972 Thermal Diffusivity: Calore User Sub

Scope: Aria Material

Thermal Diffusivity: Calore User Sub [{of|species|subindex} *Species*]= Calore_User_Sub [Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Calore user subroutine for the thermal diffusivity

4.1.973 Thermodynamic Pressure

Scope: Aria Material

Thermodynamic Pressure [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.974 Thermodynamic Pressure: Copied

Scope: Aria Material

Thermodynamic Pressure: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.975 Thermodynamic Pressure: User Plugin

Scope: Aria Material

Thermodynamic Pressure: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.976 Tic Drag: Basic

Scope: Aria Material

Tic Drag: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Model for drag created in the Theory of Interacting Continua

4.1.977 Tic Drag Coeff

Scope: Aria Material

Tic Drag Coeff [{of | species | subindex} *Species*]= Constant [Alpha_Tic = *alpha_TIC*]

Parameter	Value	Default
Species	string	undefined
alpha_TIC	real	undefined

Summary Constant value

4.1.978 Tic Drag Coeff: Copied

Scope: Aria Material

Tic Drag Coeff: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.979 Tic Drag Coeff: User Plugin

Scope: Aria Material

Tic Drag Coeff: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.980 Tortuosity Factor

Scope: Aria Material

Tortuosity Factor [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.981 Tortuosity Factor: Copied

Scope: Aria Material

Tortuosity Factor: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.982 Tortuosity Factor: User Plugin

Scope: Aria Material

Tortuosity Factor: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.983 Tortuosity Factor: Bruggeman

Scope: Aria Material

Tortuosity Factor: Bruggeman [{of | species | subindex} *Species*]= Bruggeman [Exponent = *exponent* | Multiplier = *multiplier* | Omit_Saturation = *omit_saturation*]

Parameter	Value	Default
Species	string	undefined
exponent	real	undefined
multiplier	real	undefined
omit_saturation	integer	undefined

Summary Bruggeman model for tortuosity factor. $\tau = \frac{1}{a\phi^n}$ with a default value of $n = 0.5$ and $a = 1$. Set alternate values for them with the 'exponent' and 'multiplier' arguments, respectively.

4.1.984 Tortuosity Factor: Lanfrey

Scope: Aria Material

Tortuosity Factor: Lanfrey [{of | species | subindex} *Species*]= Lanfrey [Sphericity = *sphericity* | Min_Porosity = *min_porosity*]

Parameter	Value	Default
Species	string	undefined
sphericity	real	undefined
min_porosity	real	undefined

Summary Lanfrey model for tortuosity factor - from Chemical Engineering Science 65 (2010) 1891-1896 Equation 14. $\tau = \frac{1.23}{a^2}\frac{(1-\phi)^{4/3}}{\phi}$, where a is the sphericity.

4.1.985 Tortuosity Factor: Comiti

Scope: Aria Material

Tortuosity Factor: Comiti [{of | species | subindex} *Species*]= Comiti [Min_Porosity = *min_porosity*]

Parameter	Value	Default
Species	string	undefined
min_porosity	real	undefined

Summary Comiti model for tortuosity factor - from Chem. Eng. Sci. 44, 1539-1545. $\tau = 1.0 - 0.41\ln(\phi)$

4.1.986 Total Internal Energy

Scope: Aria Material

Total Internal Energy [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.987 Total Internal Energy: Copied

Scope: Aria Material

Total Internal Energy: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.988 Total Internal Energy: User Plugin

Scope: Aria Material

Total Internal Energy: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.989 Total Internal Energy: Porous Flow

Scope: Aria Material

Total Internal Energy: Porous Flow [{of | species | subindex} *Species*]= Porous_Flow [ ]

Parameter	Value	Default
Species	string	undefined

Summary Total internal energy for porous flow

4.1.990 Transition Reynolds Number: Correlation

Scope: Aria Material

Transition Reynolds Number: Correlation [{of | species | subindex} *Species*]= Correlation [Rex = *Rex* | Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
Rex	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Reynolds number computed from material properties

4.1.991 Transport Cross Section

Scope: Aria Material

`Transport Cross Section [ {of | species | subindex} Species ]= Constant [ Trans = trans ]`

Parameter	Value	Default
Species	string	undefined
trans	real	undefined

Summary Constant value

4.1.992 Transport Cross Section: Copied

Scope: Aria Material

`Transport Cross Section: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.993 Transport Cross Section: User Plugin

Scope: Aria Material

`Transport Cross Section: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.994 Transport Cross Section: Linearized

Scope: Aria Material

`Transport Cross Section: Linearized [ {of | species | subindex} Species ]= Linearized [ Sigma_0 = sigma_0 | D_Sigma_D_Rho = d_sigma_d_rho | D_Sigma_D_T = d_sigma_d_t | T_Sigma_0 = t_sigma_0 | Rho_Sigma_0 = rho_sigma_0 ]`

Parameter	Value	Default
Species	string	undefined
sigma_0	real	undefined
d_sigma_d_rho	real	undefined
d_sigma_d_t	real	undefined
t_sigma_0	real	undefined
rho_sigma_0	real	undefined

Summary Linearized transport cross section

4.1.995 Transported Enthalpy: Standard

Scope: Aria Material

`Transported Enthalpy: Standard [ {of | species | subindex} *Species* ]= Standard [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Transported quantity for porous enthalpy equation

4.1.996 Transported Enthalpy: Porous

Scope: Aria Material

`Transported Enthalpy: Porous [ {of | species | subindex} *Species* ]= Porous [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Transported quantity for porous enthalpy equation

4.1.997 Turbulence Dissipation Rate

Scope: Aria Material

`Turbulence Dissipation Rate [ {of | species | subindex} *Species* ]= Constant [ Value = *value* ]`

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.998 Turbulence Dissipation Rate: Copied

Scope: Aria Material

`Turbulence Dissipation Rate: Copied [ {of | species | subindex} *Species* ]= Copied [ Source = *source* ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.999 Turbulence Dissipation Rate: User Plugin

Scope: Aria Material

Turbulence Dissipation Rate: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1000 Turbulence Dissipation Rate Diffusive Flux: Basic

Scope: Aria Material

Turbulence Dissipation Rate Diffusive Flux: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Basic model of turbulence dissipation rate diffusive flux

4.1.1001 Turbulent Bulk Viscosity

Scope: Aria Material

Turbulent Bulk Viscosity [{of | species | subindex} *Species*]= Constant [Kappa = *kappa*]

Parameter	Value	Default
Species	string	undefined
kappa	real	undefined

Summary Constant value

4.1.1002 Turbulent Bulk Viscosity: Copied

Scope: Aria Material

Turbulent Bulk Viscosity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1003 Turbulent Bulk Viscosity: User Plugin

Scope: Aria Material

Turbulent Bulk Viscosity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1004 Turbulent Correlation Heat Transfer Coefficient: Correlation

Scope: Aria Material

Turbulent Correlation Heat Transfer Coefficient: Correlation [{of | species | subindex} *Species*]= Correlation [Number = *number* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
number	integer	undefined
bulk_node_interface	"string"	undefined

Summary Correlation based heat transfer coefficient

4.1.1005 Turbulent Energy Diffusive Flux: Gradient Transport

Scope: Aria Material

Turbulent Energy Diffusive Flux: Gradient Transport [{of | species | subindex} *Species*]= Gradient_Transport [ ]

Parameter	Value	Default
Species	string	undefined

Summary Gradient transport model of turbulent energy diffusive flux

4.1.1006 Turbulent Kinetic Energy

Scope: Aria Material

Turbulent Kinetic Energy [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.1007 Turbulent Kinetic Energy: Copied

Scope: Aria Material

Turbulent Kinetic Energy: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1008 Turbulent Kinetic Energy: User Plugin

Scope: Aria Material

Turbulent Kinetic Energy: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1009 Turbulent Kinetic Energy Density

Scope: Aria Material

Turbulent Kinetic Energy Density [{of|species|subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.1010 Turbulent Kinetic Energy Density: Copied

Scope: Aria Material

Turbulent Kinetic Energy Density: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1011 Turbulent Kinetic Energy Density: User Plugin

Scope: Aria Material

Turbulent Kinetic Energy Density: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1012 Turbulent Kinetic Energy Diffusive Flux: Basic

Scope: Aria Material

Turbulent Kinetic Energy Diffusive Flux: Basic [{of | species | subindex} *Species*]= Basic [ ]

Parameter	Value	Default
Species	string	undefined

Summary Basic model of turbulent kinetic energy diffusive flux

4.1.1013 Turbulent Mass Diffusivity

Scope: Aria Material

Turbulent Mass Diffusivity [{of | species | subindex} *Species*]= Constant [Dt = *Dt*]

Parameter	Value	Default
Species	string	undefined
Dt	real	undefined

Summary Constant value

4.1.1014 Turbulent Mass Diffusivity: Copied

Scope: Aria Material

Turbulent Mass Diffusivity: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1015 Turbulent Mass Diffusivity: User Plugin

Scope: Aria Material

Turbulent Mass Diffusivity: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1016 Turbulent Mass Diffusivity: From Schmidt

Scope: Aria Material

Turbulent Mass Diffusivity: From Schmidt [{of|species|subindex} *Species*]= From_Schmidt [ ]

Parameter	Value	Default
Species	string	undefined

Summary Turbulent mass diffusivity from Schmidt number and viscosity

4.1.1017 Turbulent Mass Fraction Diffusive Flux: Gradient Transport

Scope: Aria Material

Turbulent Mass Fraction Diffusive Flux: Gradient Transport [{of|species|subindex} *Species*]= Gradient_Transport [ ]

Parameter	Value	Default
Species	string	undefined

Summary Gradient transport model of turbulent mass fraction diffusive flux

4.1.1018 Turbulent Mixture Fraction Diffusive Flux: Gradient Transport

Scope: Aria Material

Turbulent Mixture Fraction Diffusive Flux: Gradient Transport [{of | species | subindex} *Species*]= Gradient_Transport [ ]

Parameter	Value	Default
Species	string	undefined

Summary Gradient transport model of turbulent mixture fraction diffusive flux

4.1.1019 Turbulent Mixture Fraction Diffusivity

Scope: Aria Material

Turbulent Mixture Fraction Diffusivity [{of | species | subindex} *Species*]= Constant [Dt = *Dt*]

Parameter	Value	Default
Species	string	undefined
Dt	real	undefined

Summary Constant value

4.1.1020 Turbulent Mixture Fraction Diffusivity: Copied

Scope: Aria Material

Turbulent Mixture Fraction Diffusivity: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1021 Turbulent Mixture Fraction Diffusivity: User Plugin

Scope: Aria Material

Turbulent Mixture Fraction Diffusivity: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1022 Turbulent Mixture Fraction Diffusivity: From Schmidt

Scope: Aria Material

Turbulent Mixture Fraction Diffusivity: From Schmidt [{of | species | subindex} *Species*]= From_Schmidt [ ]

Parameter	Value	Default
Species	string	undefined

Summary Turbulent mixture fraction diffusivity from trb Schmidt number and trb viscosity

4.1.1023 Turbulent Momentum Stress: Formal Newtonian Isotropic

Scope: Aria Material

Turbulent Momentum Stress: Formal Newtonian Isotropic [{of | species | subindex} *Species*]= Formal_Newtonian_Isotropic [Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.1024 Turbulent Momentum Stress: Incompressible Newtonian Isotropic

Scope: Aria Material

Turbulent Momentum Stress: Incompressible Newtonian Isotropic [{of | species | subindex} *Species*]= Incompressible_Newtonian_Isotropic [Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.1025 Turbulent Momentum Stress: Newtonian Dilational Isotropic

Scope: Aria Material

Turbulent Momentum Stress: Newtonian Dilational Isotropic [{of | species | subindex} *Species*]= Newtonian_Dilational_Isotropic [Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.1026 Turbulent Momentum Stress: Newtonian Isotropic

Scope: Aria Material

Turbulent Momentum Stress: Newtonian Isotropic [{of | species | subindex} *Species*]= Newtonian_Isotro
[Reference_Frame = *reference_frame*]

Parameter	Value	Default
Species	string	undefined
reference_frame	"string"	undefined

4.1.1027 Turbulent Prandtl Number

Scope: Aria Material

`Turbulent Prandtl Number [ {of|species|subindex} Species ]= Constant [ Pr_T = Pr_t ]`

Parameter	Value	Default
Species	string	undefined
Pr_t	real	undefined

Summary Constant value

4.1.1028 Turbulent Prandtl Number: Copied

Scope: Aria Material

`Turbulent Prandtl Number: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1029 Turbulent Prandtl Number: User Plugin

Scope: Aria Material

`Turbulent Prandtl Number: User Plugin [ {of|species|subindex} Species ]= User_Plugin [ Name = Name |magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1030 Turbulent Schmidt Number

Scope: Aria Material

`Turbulent Schmidt Number [ {of|species|subindex} Species ]= Constant [ Sc_T = Sc_t ]`

Parameter	Value	Default
Species	string	undefined
Sc_t	real	undefined

Summary Constant value

4.1.1031 Turbulent Schmidt Number: Copied

Scope: Aria Material

Turbulent Schmidt Number: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1032 Turbulent Schmidt Number: User Plugin

Scope: Aria Material

Turbulent Schmidt Number: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1033 Turbulent Thermal Conductivity: From Prandtl

Scope: Aria Material

Turbulent Thermal Conductivity: From Prandtl [{of | species | subindex} *Species*]= From_Prandtl [ ]

Parameter	Value	Default
Species	string	undefined

Summary Turbulent thermal conductivity when closing energy term by grad T

4.1.1034 Turbulent Thermal Diffusivity: From Prandtl

Scope: Aria Material

Turbulent Thermal Diffusivity: From Prandtl [{of | species | subindex} *Species*]= From_Prandtl [ ]

Parameter	Value	Default
Species	string	undefined

Summary ?

4.1.1035 Use Data Block

Scope: Aria Material

`Use Data Block` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

4.1.1036 Valence

Scope: Aria Material

`Valence [ {of | species | subindex}` *`Species`* `]= Constant [ Z =` *`z`* `]`

Parameter	Value	Default
Species	`string`	`undefined`
z	`real`	`undefined`

Summary Constant value

4.1.1037 Valence: Copied

Scope: Aria Material

`Valence: Copied [ {of | species | subindex}` *`Species`* `]= Copied [ Source =` *`source`* `]`

Parameter	Value	Default
Species	`string`	`undefined`
source	`"string"`	`undefined`

Summary Copied value from another expression

4.1.1038 Valence: User Plugin

Scope: Aria Material

`Valence: User Plugin [ {of | species | subindex}` *`Species`* `]= User_Plugin [ Name =` *`Name`* `| magic_trailing_`
`]`

Parameter	Value	Default
Species	`string`	`undefined`
Name	`"string"`	`undefined`
magic_trailing_string_vector_parm	`"string"`	`undefined`

Summary Value from a user plugin

4.1.1039 Velocity: Melting Capillary Darcy

Scope: Aria Material

Velocity: Melting Capillary Darcy [{of|species|subindex} *Species*]= Melting_Capillary_Darcy [Melt_Dt = *melt_DT* |Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
melt_DT	real	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

4.1.1040 Velocity: Darcy

Scope: Aria Material

Velocity: Darcy [{of|species|subindex} *Species*]= Darcy [Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

4.1.1041 Velocity: Darcy Solvent

Scope: Aria Material

Velocity: Darcy Solvent [{of|species|subindex} *Species*]= Darcy_Solvent [Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

Summary This model is intended for problems where the porous species equation is being solved for pressure with a solvent species concentration as the mass term in order to solve for the velocity of that solvent species.

4.1.1042 Velocity: Schloegl

Scope: Aria Material

Velocity: Schloegl [{of|species|subindex} *Species*]= Schloegl [K_Phi = *k_phi* |F = *f* |Gx = *gx* |Gy = *gy* |Gz = *gz* |Proton_Name = *proton_name*]

Parameter	Value	Default
Species	string	undefined
k_phi	real	undefined
f	real	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined
proton_name	"string"	undefined

4.1.1043 Velocity: From Mesh Displacement

Scope: Aria Material

`Velocity: From Mesh Displacement [ {of | species | subindex} Species ]= From_Mesh_Displacement [ ]`

Parameter	Value	Default
Species	string	undefined

4.1.1044 Velocity

Scope: Aria Material

`Velocity [ {of | species | subindex} Species ]= Constant [ X = x | Y = y | Z = z ]`

Parameter	Value	Default
Species	string	undefined
x	real	undefined
y	real	undefined
z	real	undefined

Summary Constant value

4.1.1045 Velocity: User Plugin

Scope: Aria Material

`Velocity: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1046 Venting Volumetric Flow Rate

Scope: Aria Material

Venting Volumetric Flow Rate [{of | species | subindex} *Species*]= Constant [Value = *value*]

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.1047 Venting Volumetric Flow Rate: Copied

Scope: Aria Material

Venting Volumetric Flow Rate: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1048 Venting Volumetric Flow Rate: User Plugin

Scope: Aria Material

Venting Volumetric Flow Rate: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1049 Venting Volumetric Flow Rate: K Factor

Scope: Aria Material

Venting Volumetric Flow Rate: K Factor [{of | species | subindex} *Species*]= K_Factor [Flow_Rate_Lim = *flow_rate_limiter*]

Parameter	Value	Default
Species	string	undefined
flow_rate_limiter	real	undefined

Summary Volumetric flow rate, Q, calculated as $Q = K\sqrt{\frac{P-P_{ambient}}{\rho}}$.Absent any other sources, the maximum flow rate is defined based on the mass loss to achieve pressure equilibrium and the fluid time step. If the optional flow rate limiter is specified, the flow rate is limited by the product of the limiter factor and the maximum flow rate without any other sources.

4.1.1050 Venting Volumetric Flow Rate: K Factor With Choking

Scope: Aria Material

Venting Volumetric Flow Rate: K Factor With Choking [{of|species|subindex} *Species*]= K_Factor_With_Choking [Critical_Pressure_Ratio = *critical_pressure_ratio* |Flow_Rate_Limiter = *flow_rate_limiter*]

Parameter	Value	Default
Species	string	undefined
critical_pressure_ratio	real	undefined
flow_rate_limiter	real	undefined

Summary Volumetric flow rate, Q, calculated as $Q = K\sqrt{\frac{P-P_{ambient}}{\rho}}$ for $\frac{P}{P_{ambient}} < R_p$, where R_p is the critical_pressure_ratio. Above R_p the flow rate is calculated as $Q = K\sqrt{\frac{P_{high}(1-1/R_p)}{\rho}}$. Absent any other sources, the maximum flow rate is defined based on the mass loss to achieve pressure equilibrium and the fluid time step. If the optional flow rate limiter is specified, the flow rate is limited by the product of the limiter factor and the maximum flow rate without any other sources.

4.1.1051 Viscosity

Scope: Aria Material

Viscosity [{of|species|subindex} *Species*]= Constant [Mu = *mu*]

Parameter	Value	Default
Species	string	undefined
mu	real	undefined

Summary Constant value

4.1.1052 Viscosity: Copied

Scope: Aria Material

Viscosity: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1053 Viscosity: User Plugin

Scope: Aria Material

Viscosity: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_parser	"string"	undefined

Summary Value from a user plugin

4.1.1054 Viscosity: Arrhenius

Scope: Aria Material

Viscosity: Arrhenius [{of|species|subindex} *Species*]= Arrhenius [Mu0 = *mu0* |E = *e* |Mu_Max = *mu_max*]

Parameter	Value	Default
Species	string	undefined
mu0	real	undefined
e	real	undefined
mu_max	real	undefined

4.1.1055 Viscosity: Arrhenius Carreau

Scope: Aria Material

Viscosity: Arrhenius Carreau [{of|species|subindex} *Species*]= Arrhenius_Carreau [Mu_Zero = *mu_zero* |Mu_Inf = *mu_inf* |N = *n* |A = *a* |Lambda = *lambda* |K = *k* |T_Ref = *t_ref* |Skip_Sensitivitie = *skip_sensitivities* |Sensitivity_Scaling = *sensitivity_scaling* |Ramp_Sensitivities_Over_Time = *ramp_sensitivities_over_time*]

Parameter	Value	Default
Species	string	undefined
mu_zero	real	undefined
mu_inf	real	undefined
n	real	undefined
a	real	undefined
lambda	real	undefined
k	real	undefined
t_ref	real	undefined
skip_sensitivities	integer	undefined
sensitivity_scaling	real	undefined
ramp_sensitivities_over_time	real	undefined

4.1.1056 Viscosity: Bingham Wlf

Scope: Aria Material

Viscosity: Bingham Wlf [{of|species|subindex} *Species*]= Bingham_Wlf [Mu_Zero = *mu_zero* |Mu_Inf = *mu_inf* |F = *f* |N = *n* |A = *a* |Lambda = *lambda* |Tau_Y = *tau_y* |Skip_Sensitivities = *skip_sensitivities*]

Parameter	Value	Default
Species	string	undefined
mu_zero	real	undefined
mu_inf	real	undefined
f	real	undefined
n	real	undefined
a	real	undefined
lambda	real	undefined
tau_y	real	undefined
skip_sensitivities	integer	undefined

4.1.1057 Viscosity: Bingham Wlft

Scope: Aria Material

Viscosity: Bingham Wlft [{of|species|subindex} *Species*]= Bingham_Wlft [Mu_Zero = *mu_zero* |Mu_Inf = *mu_inf* |F = *f* |N = *n* |A = *a* |Lambda = *lambda* |Tau_Y = *tau_y* |T_Ref = *t_ref* |C1 = *c1* |C2 = *c2* |Skip_Sensitivities = *skip_sensitivities*]

Parameter	Value	Default
Species	string	undefined
mu_zero	real	undefined
mu_inf	real	undefined
f	real	undefined
n	real	undefined
a	real	undefined
lambda	real	undefined
tau_y	real	undefined
t_ref	real	undefined
c1	real	undefined
c2	real	undefined
skip_sensitivities	integer	undefined

4.1.1058 Viscosity: Carreau

Scope: Aria Material

Viscosity: Carreau [{of|species|subindex} *Species*]= Carreau [Mu_Zero = *mu_zero* |Mu_Inf = *mu_inf* |N = *n* |A = *a* |Lambda = *lambda* |Skip_Sensitivities = *skip_sensitivities*]

Parameter	Value	Default
Species	string	undefined
mu_zero	real	undefined
mu_inf	real	undefined
n	real	undefined
a	real	undefined
lambda	real	undefined
skip_sensitivities	integer	undefined

4.1.1059 Viscosity: Carreau T

Scope: Aria Material

Viscosity: Carreau T [{of|species|subindex} *Species*]= Carreau_T [Mu_Zero = *mu_zero* |Mu_Inf = *mu_inf* |N = *n* |A = *a* |K = *k* |Skip_Sensitivities = *skip_sensitivities*]

Parameter	Value	Default
Species	string	undefined
mu_zero	real	undefined
mu_inf	real	undefined
n	real	undefined
a	real	undefined
k	real	undefined
skip_sensitivities	integer	undefined

4.1.1060 Viscosity: Casson

Scope: Aria Material

Viscosity: Casson [{of|species|subindex} *Species*]= Casson [Mu_Zero = *mu_zero* |M = *m* |Shear_Reference = *shear_reference* |Yield_Stress = *yield_stress*]

Parameter	Value	Default
Species	string	undefined
mu_zero	real	undefined
m	real	undefined
shear_reference	real	undefined
yield_stress	real	undefined

4.1.1061 Viscosity: Cantera

Scope: Aria Material

Viscosity: Cantera [{of|species|subindex} *Species*]= Cantera [Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
poffset	real	undefined

4.1.1062 Viscosity: Clsm

Scope: Aria Material

Viscosity: Clsm [{of|species|subindex} *Species*]= Clsm [Primaryproperty = *primaryProperty* |Secondaryproperty = *secondaryProperty*]

Parameter	Value	Default
Species	string	undefined
primaryProperty	real	undefined
secondaryProperty	real	undefined

Summary CLSM viscosity

4.1.1063 Viscosity: Curing Epoxy

Scope: Aria Material

Viscosity: Curing Epoxy [{of|species|subindex} *Species*]= Curing_Epoxy [Mu_A = *mu_a* |T_Mu = *t_mu* |N_Mu = *n_mu* |Ksi_C = *ksi_c* |M_Ksi_C = *m_ksi_c* |E_Mu = *e_mu* |R = *r* |Epoxy_Subindex = *epoxy_subindex* |Epoxy_Species_Name = *epoxy_species_name* |Epoxy_Species_Phase = *epoxy_species_phase*]

Parameter	Value	Default
Species	string	undefined
mu_a	real	undefined
t_mu	real	undefined
n_mu	real	undefined
ksi_c	real	undefined
m_ksi_c	real	undefined
e_mu	real	undefined
r	real	undefined
epoxy_subindex	integer	undefined
epoxy_species_name	"string"	undefined
epoxy_species_phase	"string"	undefined

4.1.1064 Viscosity: Curing Foam

Scope: Aria Material

Viscosity: Curing Foam [{of|species|subindex} *Species*]= Curing_Foam [Extent_Subindex = *extent_subindex* |Mu_A = *mu_a* |T_Mu = *T_mu* |N_Mu = *n_mu* |Ksi_C = *ksi_c* |M_Ksi_C = *m_ksi_c* |E_Mu = *E_mu* |R = *r*]

Parameter	Value	Default
Species	string	undefined
extent_subindex	integer	undefined
mu_a	real	undefined
T_mu	real	undefined
n_mu	real	undefined
ksi_c	real	undefined
m_ksi_c	real	undefined
E_mu	real	undefined
r	real	undefined

4.1.1065 Viscosity: Keyes

Scope: Aria Material

Viscosity: Keyes [{of|species|subindex} *Species*]= Keyes [A0 = *a0* |A = *a* |A1 = *a1*]

Parameter	Value	Default
Species	string	undefined
a0	real	undefined
a	real	undefined
a1	real	undefined

4.1.1066 Viscosity: Krieger

Scope: Aria Material

Viscosity: Krieger [{of|species|subindex} *Species*]= Krieger [Beta = *beta* |Phi_Max = *phi_max* |Mu_S = *mu_s* |Use_Fd = *use_fd*]

Parameter	Value	Default
Species	string	undefined
beta	real	undefined
phi_max	real	undefined
mu_s	real	undefined
use_fd	integer	undefined

4.1.1067 Viscosity: Mixture Fraction

Scope: Aria Material

Viscosity: Mixture Fraction [{of|species|subindex} *Species*]= Mixture_Fraction [Primaryproperty = *primaryProperty* |Secondaryproperty = *secondaryProperty*]

Parameter	Value	Default
Species	string	undefined
primaryProperty	real	undefined
secondaryProperty	real	undefined

4.1.1068 Viscosity: Mixture Fraction Turbulent

Scope: Aria Material

Viscosity: Mixture Fraction Turbulent [{of|species|subindex} *Species*]= Mixture_Fraction_Turbulen
[Primaryproperty = *primaryProperty* |Secondaryproperty = *secondaryProperty*]

Parameter	Value	Default
Species	string	undefined
primaryProperty	real	undefined
secondaryProperty	real	undefined

4.1.1069 Viscosity: Morris Boulay

Scope: Aria Material

Viscosity: Morris Boulay [{of|species|subindex} *Species*]= Morris_Boulay [K_S = *K_s* |Phi_Max = *phi_max* |Phi_Tol = *phi_tol* |Mu_S = *mu_s*]

Parameter	Value	Default
Species	string	undefined
K_s	real	undefined
phi_max	real	undefined
phi_tol	real	undefined
mu_s	real	undefined

4.1.1070 Viscosity: Phase Average

Scope: Aria Material

Viscosity: Phase Average [{of | species | subindex} *Species*]= Phase_Average [Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.1071 Viscosity: Interpolated Phase Average

Scope: Aria Material

Viscosity: Interpolated Phase Average [{of | species | subindex} *Species*]= Interpolated_Phase_Averag
[Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.1072 Viscosity: Power Law

Scope: Aria Material

Viscosity: Power Law [{of | species | subindex} *Species*]= Power_Law [K = *k* |N = *n*]

Parameter	Value	Default
Species	string	undefined
k	real	undefined
n	real	undefined

4.1.1073 Viscosity: Sutherland

Scope: Aria Material

Viscosity: Sutherland [{of | species | subindex} *Species*]= Sutherland [Tref = *tref* |C = *c*]

Parameter	Value	Default
Species	string	undefined
tref	real	undefined
c	real	undefined

4.1.1074 Viscosity: Weld

Scope: Aria Material

Viscosity: Weld [{of|species|subindex} *Species*]= Weld [Interpolated = *interpolated* |Beta = *beta* |C0 = *c0* |C1 = *c1* |C2 = *c2* |C3 = *c3* |T_Liq = *T_liq* |T_90 = *t_90* |T_Max = *T_max*]

Parameter	Value	Default
Species	string	undefined
interpolated	integer	undefined
beta	real	undefined
c0	real	undefined
c1	real	undefined
c2	real	undefined
c3	real	undefined
T_liq	real	undefined
t_90	real	undefined
T_max	real	undefined

4.1.1075 Viscosity: Ramaccioti

Scope: Aria Material

Viscosity: Ramaccioti [{of|species|subindex} *Species*]= Ramaccioti [Interpolated = *interpolated* |T_Liq = *T_liq* |T_Sol = *T_sol* |C = *c* |Mu_Liq = *mu_liq*]

Parameter	Value	Default
Species	string	undefined
interpolated	integer	undefined
T_liq	real	undefined
T_sol	real	undefined
c	real	undefined
mu_liq	real	undefined

4.1.1076 Viscosity: Thermal

Scope: Aria Material

Viscosity: Thermal [{of|species|subindex} *Species*]= Thermal [A = *a* |B = *b* |C = *c* |D = *d*]

Parameter	Value	Default
Species	string	undefined
a	real	undefined
b	real	undefined
c	real	undefined
d	real	undefined

4.1.1077 Viscosity: Mass Average

Scope: Aria Material

Viscosity: Mass Average [{of | species | subindex} *Species*]= Mass_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Viscosity computed from mass average of species viscosities

4.1.1078 Viscosity: Species Average

Scope: Aria Material

Viscosity: Species Average [{of | species | subindex} *Species*]= Species_Average [ ]

Parameter	Value	Default
Species	string	undefined

Summary Viscosity computed from mass average of species viscosities

4.1.1079 Voltage

Scope: Aria Material

Voltage [{of | species | subindex} *Species*]= Constant [Toggle = *toggle* | Value = *value*]

Parameter	Value	Default
Species	string	undefined
toggle	"string"	undefined
value	real	undefined

Summary Constant value

4.1.1080 Voltage: Copied

Scope: Aria Material

Voltage: Copied [{of | species | subindex} *Species*]= Copied [Toggle = *toggle* | Source = *source*]

Parameter	Value	Default
Species	string	undefined
toggle	"string"	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1081 Voltage: User Plugin

Scope: Aria Material

Voltage: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Toggle = *toggle* | Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
toggle	"string"	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1082 Volume Fraction

Scope: Aria Material

`Volume Fraction [ {of | species | subindex} Species ]= Constant [ Value = Value ]`

Parameter	Value	Default
Species	string	undefined
Value	real	undefined

Summary Constant value

4.1.1083 Volume Fraction: Copied

Scope: Aria Material

`Volume Fraction: Copied [ {of | species | subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1084 Volume Fraction: User Plugin

Scope: Aria Material

`Volume Fraction: User Plugin [ {of | species | subindex} Species ]= User_Plugin [ Name = Name | magic_trailing_string_vector_param ]`

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1085 Volume Fraction: From Mass Fraction

Scope: Aria Material

Volume Fraction: From Mass Fraction [{of|species|subindex} *Species*]= From_Mass_Fraction [ ]

Parameter	Value	Default
Species	string	undefined

Summary Compute the volume fraction from the ratio of species porous density (rho_bulk * Y_i) to species solid density.The overall density material phase to use can be specified with the optional 'Density_Phase_Name' argument, which canbe necessary for porous systems depending on how the mass fractions are defined.

4.1.1086 Volume Fraction: From Porosity

Scope: Aria Material

Volume Fraction: From Porosity [{of|species|subindex} *Species*]= From_Porosity [ ]

Parameter	Value	Default
Species	string	undefined

Summary Compute the volume fraction from a species porosity.

4.1.1087 Volume Fraction: From Molar Volume

Scope: Aria Material

Volume Fraction: From Molar Volume [{of|species|subindex} *Species*]= From_Molar_Volume [ ]

Parameter	Value	Default
Species	string	undefined

Summary Compute the volume fraction of a species as the product of it's concentration and partial molar volume.

4.1.1088 Volume Fraction: From Electrode Object

Scope: Aria Material

Volume Fraction: From Electrode Object [{of|species|subindex} *Species*]= From_Electrode_Object [Electrodefile = *electrodeFile* |F = *f* |Kmolconversion = *kmolConversion* |Jconversion = *JConversion* |Meterconversion = *meterConversion* |Mincapacity = *minCapacity* |Deactivationvoltage = *deactivationVolta* |Voltagevariablename = *voltageVariableName* |Abortaction = *abortAction* |Inactivesolidfraction = *inactiveSolidFraction* |Pref = *Pref* |Sens_Eps = *Sens_eps* |Sens_Order = *Sens_order* |Max_Fraction = *max_fraction*]

Parameter	Value	Default
Species	string	undefined
electrodeFile	"string"	undefined
f	real	undefined
kmolConversion	real	undefined
JConversion	real	undefined
meterConversion	real	undefined
minCapacity	real	undefined
deactivationVoltage	real	undefined
voltageVariableName	"string"	undefined
abortAction	"string"	undefined
inactiveSolidFraction	real	undefined
Pref	real	undefined
Sens_eps	real	undefined
Sens_order	"string"	undefined
max_fraction	real	undefined

Summary Get the solid volume fraction for all species tracked by the Electrode object being used to calculate electrochemical reaction source terms.

4.1.1089 Volume Fraction: From Electrode Object Old

Scope: Aria Material

```
Volume Fraction:  From Electrode Object Old [ {of|species|subindex} Species ]= From_Electrode_Objec
[ F = f  |Electrodefile = electrodeFile  |Kmolconversion = kmolConversion  |Jconversion = JConversion
|Meterconversion = meterConversion  |Max_Sub_Timesteps = max_sub_timesteps  |Base_Delta = base_delta
]
```

Parameter	Value	Default
Species	string	undefined
f	real	undefined
electrodeFile	"string"	undefined
kmolConversion	real	undefined
JConversion	real	undefined
meterConversion	real	undefined
max_sub_timesteps	integer	undefined
base_delta	real	undefined

Summary Get the solid volume fraction for all species tracked by the Electrode object being used to calculate electrochemical reaction source terms.

4.1.1090 Volume Fraction Gas

Scope: Aria Material

```
Volume Fraction Gas [ {of|species|subindex} Species ]= Constant [ Value = value ]
```

Parameter	Value	Default
Species	string	undefined
value	real	undefined

Summary Constant value

4.1.1091 Volume Fraction Gas: Copied

Scope: Aria Material

Volume Fraction Gas: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1092 Volume Fraction Gas: User Plugin

Scope: Aria Material

Volume Fraction Gas: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1093 Volume Fraction Gas: From Density

Scope: Aria Material

Volume Fraction Gas: From Density [{of | species | subindex} *Species*]= From_Density [Rho_Liquid = *rho_liquid* | Rho_Gas = *rho_gas* | Rho_Initial = *rho_initial*]

Parameter	Value	Default
Species	string	undefined
rho_liquid	real	undefined
rho_gas	real	undefined
rho_initial	real	undefined

Summary Compute the gas volume fraction from the gas density

4.1.1094 Volume Fraction Gas: Fromfoamtimetemp

Scope: Aria Material

Volume Fraction Gas: Fromfoamtimetemp [{of | species | subindex} *Species*]= Fromfoamtimetemp [Rho_Liquidfluorinert = *rho_liquidfluorinert* | Rho_Vaporfluorinert = *rho_vaporfluorinert* | Rho_Air = *rho_air* | Massfraction_Air = *massfraction_air* | Rho_Initial = *rho_initial*]

Parameter	Value	Default
Species	string	undefined
rho_liquidfluorinert	real	undefined
rho_vaporfluorinert	real	undefined
rho_air	real	undefined
massfraction_air	real	undefined
rho_initial	real	undefined

Summary Compute the gas volume fraction from the foam time/temperature profile

4.1.1095 Volume Fraction Gas: Phase Average

Scope: Aria Material

Volume Fraction Gas: Phase Average [{of|species|subindex} *Species*]= Phase_Average [Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.1096 Volume Fraction Gas: Interpolated Phase Average

Scope: Aria Material

Volume Fraction Gas: Interpolated Phase Average [{of|species|subindex} *Species*]= Interpolated_Pha [Subindex_A = *subindex_a* |Subindex_B = *subindex_b*]

Parameter	Value	Default
Species	string	undefined
subindex_a	integer	undefined
subindex_b	integer	undefined

Summary Phase-averaged values

4.1.1097 Volume Fraction Gas: From Reaction Extent

Scope: Aria Material

Volume Fraction Gas: From Reaction Extent [{of|species|subindex} *Species*]= From_Reaction_Extent [Al_1 = *al_1* |Al_2 = *al_2* |V_Liq = *v_liq* |N_I = *n_i* |N_Max = *n_max* |Mw = *mw* |T_Ref = *t_ref*]

Parameter	Value	Default
Species	string	undefined
al_1	real	undefined
al_2	real	undefined
v_liq	real	undefined
n_i	real	undefined
n_max	real	undefined
mw	real	undefined
t_ref	real	undefined

Summary Gas volume fraction from reaction extent for foaming applications

4.1.1098 Volume Fraction Gas: Species

Scope: Aria Material

Volume Fraction Gas: Species [{of | species | subindex} *Species*]= Species [Gas_Species_Subindex = *gas_species_subindex* |Gas_Species_Name = *gas_species_name*]

Parameter	Value	Default
Species	string	undefined
gas_species_subindex	integer	undefined
gas_species_name	"string"	undefined

Summary Compute the gas volume fraction as C_g * MW_g / rho_g where C_g is the molar concentration of the gas, MW_g is the molecular weight, and rho_g is the density of the pure gas at the current T, P. The species to use for C_g and MW_g can be specified either by name or by subindex.

4.1.1099 Volume Fraction Gas: From Thermal Expansion

Scope: Aria Material

Volume Fraction Gas: From Thermal Expansion [{of | species | subindex} *Species*]= From_Thermal_Expans
[Phi0 = *phi0* |Rhob0 = *rhob0* |Rhoc0 = *rhoc0* |Beta0 = *beta0* |Beta1 = *beta1* |T0 = *t0*]

Parameter	Value	Default
Species	string	undefined
phi0	real	undefined
rhob0	real	undefined
rhoc0	real	undefined
beta0	real	undefined
beta1	real	undefined
t0	real	undefined

Summary Gas volume fraction from thermal expansion of the condensed phase

4.1.1100 Volume Fraction Gas: From Mass Fractions

Scope: Aria Material

`Volume Fraction Gas: From Mass Fractions [ {of|species|subindex} Species ]= From_Mass_Fractions [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Compute the gas volume fraction from the mass fractions and bulk densities

4.1.1101 Volume Fraction Gas: From Species

Scope: Aria Material

`Volume Fraction Gas: From Species [ {of|species|subindex} Species ]= From_Species [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Compute the gas volume fraction from the species concentrations pulled from ChemEq

4.1.1102 Volume Fraction Gas: From Porosity

Scope: Aria Material

`Volume Fraction Gas: From Porosity [ {of|species|subindex} Species ]= From_Porosity [ ]`

Parameter	Value	Default
Species	string	undefined

Summary Gas volume fraction from the porosity

4.1.1103 Volumetric Heat Transfer Coefficient

Scope: Aria Material

`Volumetric Heat Transfer Coefficient [ {of|species|subindex} Species ]= Constant [ H = h ]`

Parameter	Value	Default
Species	string	undefined
h	real	undefined

Summary Constant value

4.1.1104 Volumetric Heat Transfer Coefficient: Copied

Scope: Aria Material

`Volumetric Heat Transfer Coefficient: Copied [ {of|species|subindex} Species ]= Copied [ Source = source ]`

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1105 Volumetric Heat Transfer Coefficient: User Plugin

Scope: Aria Material

Volumetric Heat Transfer Coefficient: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1106 Wall Angle: Correlation

Scope: Aria Material

Wall Angle: Correlation [{of|species|subindex} *Species*]= Correlation [Theta = *theta* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
theta	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Wall Angle

4.1.1107 Wall Length: Correlation

Scope: Aria Material

Wall Length: Correlation [{of|species|subindex} *Species*]= Correlation [L = *l* |Bulk_Node_Interfa = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
l	real	undefined
bulk_node_interface	"string"	undefined

Summary Correlation Wall length

4.1.1108 Wall Temperature

Scope: Aria Material

Wall Temperature [{of | species | subindex} *Species*]= Constant [Tw = *Tw*]

Parameter	Value	Default
Species	string	undefined
Tw	real	undefined

Summary Constant value

4.1.1109 Wall Temperature: Copied

Scope: Aria Material

Wall Temperature: Copied [{of | species | subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1110 Wall Temperature: User Plugin

Scope: Aria Material

Wall Temperature: User Plugin [{of | species | subindex} *Species*]= User_Plugin [Name = *Name* | *magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1111 Wetted Perimeter

Scope: Aria Material

Wetted Perimeter [{of | species | subindex} *Species*]= Constant [P = *p*]

Parameter	Value	Default
Species	string	undefined
p	real	undefined

Summary Constant value

4.1.1112 Wetted Perimeter: Copied

Scope: Aria Material

Wetted Perimeter: Copied [{of|species|subindex} *Species*]= Copied [Source = *source*]

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1113 Wetted Perimeter: User Plugin

Scope: Aria Material

Wetted Perimeter: User Plugin [{of|species|subindex} *Species*]= User_Plugin [Name = *Name* |*magic_trailing_string_vector_param*]

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1114 Wetted Perimeter: Spatial User Function

Scope: Aria Material

Wetted Perimeter: Spatial User Function [{of|species|subindex} *Species*]= Spatial_User_Function [Threshold = *threshold* |Multiplier = *multiplier* |Name = *name* |Bar = *bar* |X = *x*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined

Summary Spatially varying wetted perimeter based on a coordinate dependent user function

4.1.1115 Wetted Perimeter: Correlation Spatial User Function

Scope: Aria Material

Wetted Perimeter: Correlation Spatial User Function [{of|species|subindex} *Species*]= Correlation_Spatial_User_Function [Threshold = *threshold* |Multiplier = *multiplier* |Name = *name* |Bar = *bar* |X = *x* |Bulk_Node_Interface = *bulk_node_interface*]

Parameter	Value	Default
Species	string	undefined
threshold	real	undefined
multiplier	real	undefined
name	"string"	undefined
bar	"string"	undefined
x	"string"	undefined
bulk_node_interface	"string"	undefined

Summary Correlation spatially varying wetted perimeter based on a coordinate dependent user function

4.1.1116 Wetting Phase Retardation

Scope: Aria Material

```
Wetting Phase Retardation [ {of | species | subindex} Species ]= Constant [ R = r ]
```

Parameter	Value	Default
Species	string	undefined
r	real	undefined

Summary Constant value

4.1.1117 Wetting Phase Retardation: Copied

Scope: Aria Material

```
Wetting Phase Retardation: Copied [ {of | species | subindex} Species ]= Copied [ Source =
source ]
```

Parameter	Value	Default
Species	string	undefined
source	"string"	undefined

Summary Copied value from another expression

4.1.1118 Wetting Phase Retardation: User Plugin

Scope: Aria Material

```
Wetting Phase Retardation: User Plugin [ {of | species | subindex} Species ]= User_Plugin
[ Name = Name | magic_trailing_string_vector_param ]
```

Parameter	Value	Default
Species	string	undefined
Name	"string"	undefined
magic_trailing_string_vector_param	"string"	undefined

Summary Value from a user plugin

4.1.1119 Wisdom: Magic Eight Ball

Scope: Aria Material

`Wisdom: Magic Eight Ball [ {of | species | subindex} Species ]= Magic_Eight_Ball [ Question = question ]`

Parameter	Value	Default
Species	`string`	`undefined`
question	`"string"`	`undefined`

Summary The Eight ball knows all

4.2 BAR AREA

BAR AREA = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the bar cross-sectional area.

Summary Specifies the material model and parameters for the bar cross-sectional area.

Parent Block(s) `ARIA MATERIAL`

4.2.1 BAR AREA = CONSTANT

Parameters `A = REAL`

Example `Bar Area = Constant A = 0.8`

Description `A` is the value of the bar cross-sectional area.

4.2.2 BAR AREA = ELEMENT_ATTRIBUTE

Parameters `MULTIPLIER = REAL`

Example `Bar Area = Element_Attribute multiplier = 1.0`

Description For bar elements the ExodusII mesh file can contain area attribute values for each element within an element block. This value can then be retrieved and used by the application. This model is used to scale the value of area attribute by `Multiplier` to obtain the bar cross-sectional area for a particular element.

4.3 BETA

BETA = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the coefficient for thermal stress.

Summary The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk}\boldsymbol{I} + 2\mu\boldsymbol{E} - \beta\left(T - T_{ref}\right)\boldsymbol{I} \tag{4.1}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.2}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu\frac{\nu}{(1-2\nu)} \tag{4.3}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha(3\lambda + 2\mu) \tag{4.4}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Parent Block(s) `ARIA MATERIAL`

4.3.1 BETA = CONSTANT

Parameters `BETA = REAL`

Example `BETA = CONSTANT BETA = 1.0`

Description `BETA` is the value of β.

4.3.2 BETA = CONVERTED

Parameters `(None)`

Example `BETA = Converted`

Description Aria will use Young's modulus, Poisson ratio and CTE to compute the Lamé β coefficient. Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

4.3.3 BETA = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Beta = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.3.4 BETA = LINEAR

Parameters
```
A = REAL
B = REAL
```

Example
```
BETA = LINEAR A = 1.0 B = -.005
```

Description β is a linear function of temperature T,

$$\beta = A + BT \tag{4.5}$$

4.4 BULK VISCOSITY

BULK VISCOSITY = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the fluid bulk viscosity.

Summary Specifies the material model for the fluid bulk viscosity.

Parent Block(s) `ARIA MATERIAL`

4.4.1 BULK VISCOSITY = CONSTANT

Parameters
```
KAPPA = REAL
```

Example
```
BULK VISCOSITY = CONSTANT KAPPA = 1.0e-5
```

Description `KAPPA` is the value of the constant fluid bulk viscosity.

4.4.2 BULK VISCOSITY = CURING_FOAM

Parameters
```
VFRAC_SUBINDEX = INT
EXTENT_SUBINDEX = INT
PHI_ZERO = REAL
[A = REAL]
[B = REAL]
[C = REAL]
[KSI_C = REAL]
```

Example
```
Bulk Viscosity = Curing_Foam Vfrac_Subindex=1 Extent_Subindex=2
Phi_Zero=0.45
```

Description For a curing epoxy with volume fraction ϕ and extent of reaction ξ the viscosity is given by

$$\kappa = \frac{4}{3}\mu_\circ \frac{\phi_\circ - \phi - 1}{\phi_\circ - \phi} \tag{4.6}$$

where $\mu_\circ$ is given by

$$\mu_\circ = (a - bT)\left(\frac{\xi_c^2 - \xi^2}{\xi_c^2}\right)^c \tag{4.7}$$

where T is the temperature. The remaining parameters a, b, c and ξ_c have default values of $a = 20$, $b = 0.22$, $c = -4/3$ and $\xi_c = 0.45$ though they can be overridden with the optional model parameters.

NOTE: The volume fraction is assumed to be a `SPECIES` field with the subindex provided by the `VFRAC_SUBINDEX` parameter. Likewise, the extent of reaction field is assumed to be a `SPECIES` field with the subindex provided by the `EXTENT_SUBINDEX` parameter.

4.4.3 BULK VISCOSITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Bulk Viscosity = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.5 CTE

CTE = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the coefficient of thermal expansion.

Summary The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk}\boldsymbol{I} + 2\mu\boldsymbol{E} - \beta\left(T - T_{ref}\right)\boldsymbol{I} \tag{4.8}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.9}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu\frac{\nu}{(1-2\nu)} \tag{4.10}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha(3\lambda + 2\mu) \tag{4.11}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

Parent Block(s) `ARIA MATERIAL`

4.5.1 CTE = CONSTANT

Parameters `CTE = REAL`

Example `CTE = CONSTANT cte = 1.0`

Description `CTE` is the value of the coefficient of thermal expansion.

4.5.2 CTE = ENCORE_FUNCTION

Parameters `NAME = STRING`
`[EVAL_TYPE = STRING]`

Example `CTE = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.6 CURRENT DENSITY

CURRENT DENSITY = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material (constitutive) model for the current density in the bulk.

Summary Specifies the material (constitutive) model for the current density in the bulk.

Parent Block(s) `ARIA MATERIAL`

4.6.1 CURRENT DENSITY = BASIC

This is an alias for `OHMS_LAW`.

Example `Current Density = Basic`

4.6.2 CURRENT DENSITY = DIELECTRIC_DISPLACEMENT

Parameters `(none)`

Example `Current Density = Dielectric_Displacement`

Description The current density $\boldsymbol{J}$ is taken to be

$$\boldsymbol{J} = -\epsilon \boldsymbol{\nabla} \frac{\partial V}{\partial t} - \frac{\partial \epsilon}{\partial t} \boldsymbol{\nabla} V \tag{4.12}$$

where ϵ is the electrical permittivity and V is the voltage (electric potential). This model can be used to simulate dielectric materials adjoined to conducting materials in transient problems.

This arises from time-differentiating Gauss' Law

$$-\boldsymbol{\nabla} \cdot (\epsilon \boldsymbol{\nabla} V) = \rho_f \tag{4.13}$$

which gives

$$-\boldsymbol{\nabla} \cdot \left(\epsilon \boldsymbol{\nabla} \frac{\partial V}{\partial t} + \frac{\partial \epsilon}{\partial t} \boldsymbol{\nabla} V \right) = \frac{\partial \rho_f}{\partial t}. \tag{4.14}$$

The left hand side of this equation can be used to eliminate the $\partial \rho_f / \partial t$ from the current density equation.

4.6.3 CURRENT DENSITY = OHMS_LAW

Parameters `(none)`

Example `Current Density = Ohms_Law`

Description The current density $\boldsymbol{J}$ is given by Ohm's Law,

$$\boldsymbol{J} = -\sigma_e \boldsymbol{\nabla} V \tag{4.15}$$

where σ_e is the electrical conductivity and V is the voltage (electric potential).

4.6.4 CURRENT DENSITY = THERMOELECTRIC

Parameters `(none)`

Example `Current Density = Thermoelectric`

Description The current density $\boldsymbol{J}$ is given by a combination of Ohm's Law and a contribution related to the temperature gradient,

$$\boldsymbol{J} = -\sigma_e (\boldsymbol{\nabla} V + \alpha \boldsymbol{\nabla} T) \tag{4.16}$$

where σ_e is the electrical conductivity, V is the voltage (electric potential), α is the Seebeck coefficient and T is the temperature. This current density is used in conjunction with a THERMOELECTRIC heat conduction 4.15.5.

4.7 DENSITY

DENSITY = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the density.

Summary Specifies the material model for the density.

Parent Block(s) `ARIA MATERIAL`

4.7.1 DENSITY = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example
```
Density = Calore_User_Sub name=w80afftuser type=element NR=2 R0=1000
R1=0.7
```

Description `NAME` is the name of the user subroutine was registered with. `NR` and `NI` are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the `R`n (`I`n) parameter. **Note:** the parameters use zero based counting. The `TYPE` parameter denotes the API the user subroutine uses; currently Aria only supports types `ELEMENT` and `NODE`.

The optional `MULTIPLIER` parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

4.7.2 DENSITY = COMPRESSIBLE_BOUSSINESQ

Parameters
```
REF_DENSITY
ALPHA = REAL
REF_PRESSURE
[BETA = REAL]
[REF_TEMPERATURE]
```

Example
```
Density = Compressible_Boussinesq Ref_Density=1.0 alpha=1e-8
Ref_Pressure=101235 beta=1e-3 Ref_Temperature=298
```

Description

$$\rho = \rho_{ref}\left(1 + \alpha\left(P - P_{ref}\right) + \beta\left(T - T_{ref}\right)\right) \tag{4.17}$$

Here ρ_{ref} is specified by the `REF_DENSITY` parameter and is the density at pressure $P = P_{ref}$ (given by the `REF_PRESSURE` parameter) and temperature $T = T_{ref}$ (given by the `REF_TEMPERATURE` parameter). The parameter `ALPHA` specifies the isothermal compressibility α. The parameter `BETA` specifies the isobaric compressibility β.

However, if the temperature is not available, then the temperature effects are not included and the `BETA` and `REF_TEMPERATURE` parameters do not need to be supplied.

4.7.3 DENSITY = CONSTANT

Parameters
```
RHO = REAL
```

Example
```
DENSITY = CONSTANT RHO = 1.0
```

Description `RHO` is the value of the constant density.

4.7.4 DENSITY = CURING_FOAM

Parameters
```
R = REAL
RHO_E = REAL
RHO_F = REAL
PHI_ZERO = REAL
VFRAC_SUBINDEX = INT
```

Example
```
Molecular Weight = Constant Subindex=1 M = 17.0
Density = Curing_Foam R=8.314472E3 RHO_E=1 RHO_F=1.5 PHI_ZERO=0.2
VFRAC_SUBINDEX=1
```

Description The density curing epoxy foam with volume fraction ϕ, molecular weight M, temperature T, and pressure p is given by

$$\rho = (\phi_\circ - \phi)\,\frac{pM}{RT} + (1 - \phi_\circ)\,\rho_e + \phi\rho_f \tag{4.18}$$

where R is the gas constant, $\phi_\circ$ is the reference volume fraction in the flourinert ρ_e is the pure epoxy density and ρ_f is the pure flourinert density.

NOTE: The volume fraction is assumed to be a `SPECIES` field with the subindex provided by the `VFRAC_SUBINDEX` parameter.

4.7.5 DENSITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Density = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.7.6 DENSITY = EXP_DECAY

Parameters
```
RHO_INITIAL = REAL
RHO_FINAL = REAL
K = REAL
```

Example
```
Density = Exp_Decay K=1.2 RHO_INITIAL=1.0 RHO_FINAL=0.2
```

Description This model supplies a density that is an exponential decay,

$$\rho = \rho_f + (\rho_i - \rho_f)\, e^{-kt} \tag{4.19}$$

where ρ_i is the initial density (`RHO_INITIAL`), ρ_f is the final density (`RHO_FINAL` and k (`K`) is the decay constant.

4.7.7 DENSITY = IDEAL_GAS

Parameters
```
R = REAL
[P_REF = REAL]
[T_REF = REAL]
```

Example
```
Molecular Weight = Constant Subindex=1 M = 17.0
Molecular Weight = Constant Subindex=2 M = 23.0
Density = Ideal_Gas R=8.314472E3 T_ref=273.15 P_ref=101325.0
```

Description The density of a multicomponent ideal gas in kg m^{-3} may be written as

$$\rho = \frac{P_{ref} + P}{R\,(T_{ref} + T)} \sum_{i}^{N} M_i y_i \tag{4.20}$$

where N is the number of species, P is the pressure in Pascals, P_{ref} is a reference pressure, R is the gas constant, T is the temperature, T_{ref} is a reference temperature, M_i is the molecular weight of species i in kg kmol^{-1} and y_i is the mole fraction of species i. Note, the units given here and on the density card are SI units; any units may be used as long as internal consistency is maintained.

The optional reference values for the temperature and pressure allow you to solve for the temperature and pressure using relative units (e.g., Celsius temperature and gauge pressure) and but still use absolute values as required by this material model.

4.7.8 DENSITY = INCOMPRESSIBLE_IDEAL_GAS

Parameters
```
R = REAL
P_REF = REAL
[T_REF = REAL]
```

Example

```
Molecular Weight = Constant Subindex=1 M = 17.0
Molecular Weight = Constant Subindex=2 M = 23.0
Density = Incompressible_Ideal_Gas R=8.314472E3 T_ref=273.15
P_ref=101325.0
```

Description

The density of a multicomponent ideal gas in kg m^{-3} may be written as

$$\rho = \frac{P_{ref}}{R\left(T_{ref} + T\right)} \sum_{i}^{N} M_i y_i \tag{4.21}$$

where N is the number of species, P_{ref} is a reference pressure in pascal, R is the gas constant in J kmol^{-1} K^{-1}, T is the temperature, T_{ref} is a reference temperature, M_i is the molecular weight of species i in kg kmol^{-1}, and y_i is the mole fraction of species i. Note, the units given here and on the density card are SI units; any units may be used as long as internal consistency is maintained.

The optional reference value for the temperature allow you to solve for the temperature using relative units (e.g., Celsius temperature) and but still use absolute values as required by this material model.

4.7.9 DENSITY = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Density= Polynomial Variable=Temperature Order=1 C0=401.0 C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$\rho = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.22}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.7.10 DENSITY = SINGLE_COMPONENT_IDEAL_GAS

Parameters

```
R = REAL
[M = REAL]
[P_REF = REAL]
[T_REF = REAL]
```

Example

```
Density = Single_Component_Ideal_Gas R=8.314 T_ref=273.15 P_ref=101325.0
```

Description This supplies the density for a single component ideal gas,

$$\rho = \frac{(P_{ref} + P)\, M}{R\,(T_{ref} + T)} \tag{4.23}$$

P is the pressure, P_{ref} is a reference pressure (defaults to zero), R is the gas constant, T is the temperature, T_{ref} is a reference temperature (defaults to zero), and M is the molecular weight (defaults to one).

The optional reference values for the temperature and pressure allow you to solve for the temperature and pressure using relative units (e.g., Celsius temperature and gauge pressure) and but still use absolute values as required by this material model.

4.7.11 DENSITY = THERMAL

Parameters
```
[A = REAL]
[B = REAL]
[C = REAL]
[D = REAL]
```

Example
```
DENSITY = THERMAL A = 1.0 B = -.005
```

Description Cubic polynomial function of temperature for the density.

$$\rho = A + BT + CT^2 + DT^3 \tag{4.24}$$

4.7.12 DENSITY = USER_FUNCTION

Parameters
```
NAME = STRING
X = STRING
```

Example

```
Begin Definition for Function Water_Density
   # Source Appendix 2 from "Transport Processes and
   # Unit Operations" by C. J. Geankoplis
   Type is Piecewise Linear
   Begin Values
      # K     kg * m^-3
      273.15   999.87
      277.15  1000.00
      283.15   999.73
      293.15   998.23
      298.15   997.08
      303.15   995.68
      313.15   992.25
      323.15   988.07
      333.15   983.24
      343.15   977.81
      353.15   971.83
      363.15   965.34
      373.15   958.38
   End
End
...

Begin Aria Material Foo
   ...
   Density = User_Function Name=Water_Density X=Temperature
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the density as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`Water_Density` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.8 ELECTRICAL CONDUCTIVITY

ELECTRICAL CONDUCTIVITY = ***MODEL***[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description

Specifies the material model for the electrical conductivity.

Summary

Specifies the material model for the electrical conductivity.

Parent Block(s) `ARIA MATERIAL`

4.8.1 ELECTRICAL CONDUCTIVITY = CONSTANT

Parameters
```
SIGMA = REAL
```

Example
```
ELECTRICAL CONDUCTIVITY = CONSTANT SIGMA = 1.0
```

Description `SIGMA` is the value of the constant electrical conductivity.

4.8.2 ELECTRICAL CONDUCTIVITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
ELECTRICAL CONDUCTIVITY = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.8.3 ELECTRICAL CONDUCTIVITY = EXPONENTIAL

Parameters
```
VARIABLE = STRING
[CONSTANT = REAL]
[MULTIPLIER = REAL]
EXPONENT = REAL
```

Example
```
Electrical Conductivity = Exponential Variable=Temperature Multiplier=1.0
Exponent=-0.3
```

Description Exponential function of in specified scalar variable. The electrical conductivity is computed as

$$\sigma_e = C + Me^{EX} \tag{4.25}$$

Here, C is the constant term supplied by the `CONSTANT` parameter which defaults to zero, M is the value supplied by the `MULTIPLIER` parameter which defaults to unity, X is the variable supplied by the `VARIABLE` parameter and E is the exponential multiplier provided by the `EXPONENT` parameter.

4.8.4 ELECTRICAL CONDUCTIVITY = FROM_RESISTANCE

Parameters *None.*

Example
```
ELECTRICAL CONDUCTIVITY = FROM_RESISTANCE
```

Description The conductivity is computed as the inverse of the electrical resistance which must be provided separately.

4.8.5 ELECTRICAL CONDUCTIVITY = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Electrical Conductivity= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$\sigma_e = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.26}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.8.6 ELECTRICAL CONDUCTIVITY = TBC

Parameters
```
Ki = REAL
Ti = REAL
E = REAL
R = REAL
```

Example
```
ELECTRICAL CONDUCTIVITY = TBC Ki=1.0 Ti=273 R=8.314 E=1e-3
```

Description Thermal battery electrical conductivity model (see Ken Chen).

$$\kappa(T) = \kappa_i \frac{T_i}{T} e^{-\frac{E}{R}\left(\frac{1}{T} - \frac{1}{T_i}\right)} \tag{4.27}$$

Here, T is temperature, T_i is the initial temperature provided by `Ti`, κ_i is the electrical conductivity at T_i provided by `Ki`, R is the universal gas constant provided by `R` and E is the energy provided by `E`.

4.8.7 ELECTRICAL CONDUCTIVITY = THERMAL

Parameters
```
[A = REAL]
[B = REAL]
[C = REAL]
[D = REAL]
```

Example
```
ELECTRICAL CONDUCTIVITY = THERMAL A = 1.0 B = -0.01
```

Description Cubic polynomial function of temperature for the conductivity.

$$\sigma_e = A + BT + CT^2 + DT^3 \tag{4.28}$$

4.9 ELECTRIC DISPLACEMENT

ELECTRIC DISPLACEMENT = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material (constitutive) model for the electric displacement

Summary Specifies the material (constitutive) model for the electric displacement

Parent Block(s) `ARIA MATERIAL`

4.9.1 ELECTRIC DISPLACEMENT = BASIC

This is an alias for `LINEAR`.

Example
```
Electric Displacement = Basic
```

4.9.2 ELECTRIC DISPLACEMENT = LINEAR

Parameters `(none)`

Example
```
Electric Displacement = Linear
```

Description The electric displacement $\boldsymbol{D}$ is linearly proportional to the electric field ($\boldsymbol{E} = -\boldsymbol{\nabla} V$)

$$\boldsymbol{D} = -\epsilon \boldsymbol{\nabla} V \tag{4.29}$$

where ϵ is the electrical permittivity and V is the voltage (electric potential).

4.10 ELECTRICAL PERMITTIVITY

ELECTRICAL PERMITTIVITY = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the electrical permittivity.

Summary Specifies the material model for the electrical permittivity.

Parent Block(s) `ARIA MATERIAL`

4.10.1 ELECTRICAL PERMITTIVITY = CONSTANT

Parameters
```
E = REAL
```

Example
```
ELECTRICAL PERMITTIVITY = CONSTANT E = 1.0
```

Description `E` is the value of the constant electrical permittivity.

4.10.2 ELECTRICAL PERMITTIVITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
ELECTRICAL PERMITTIVITY = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.11 ELECTRICAL RESISTANCE

ELECTRICAL RESISTANCE = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the electrical resistance.

Summary Specifies the material model for the electrical resistance.

Parent Block(s) `ARIA MATERIAL`

4.11.1 ELECTRICAL RESISTANCE = CONSTANT

Parameters *None.*

Example
```
ELECTRICAL RESISTANCE = CONSTANT R = 1.0
```

Description `R` is the value of the constant electrical resistance.

4.11.2 ELECTRICAL RESISTANCE = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
ELECTRICAL RESISTANCE = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.11.3 ELECTRICAL RESISTANCE = EXPONENTIAL

Parameters
```
VARIABLE = STRING
[CONSTANT = REAL]
[MULTIPLIER = REAL]
EXPONENT = REAL
```

Example
```
Electrical Resistance = Exponential Variable=Temperature Multiplier=1.0
Exponent=-0.3
```

Description Exponential function of in specified scalar variable. The electrical resistance is computed as

$$R = C + Me^{EX} \tag{4.30}$$

Here, C is the constant term supplied by the `CONSTANT` parameter which defaults to zero, M is the value supplied by the `MULTIPLIER` parameter which defaults to unity, X is the variable supplied by the `VARIABLE` parameter and E is the exponential multiplier provided by the `EXPONENT` parameter.

4.11.4 ELECTRICAL RESISTANCE = FROM_CONDUCTIVITY

Parameters
```
R = REAL
```

Example
```
ELECTRICAL RESISTANCE = FROM_CONDUCTIVITY
```

Description The resistance is computed as the inverse of the electrical conductivity which must be provided separately.

4.11.5 ELECTRICAL RESISTANCE = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Electrical Resistance= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$R = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.31}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.11.6 ELECTRICAL RESISTANCE = USER_FUNCTION

Parameters
```
NAME = STRING
X = STRING
```

Example

```
Begin definition for function RESISTANCE_DATA
   Type is piecewise linear
   Abscissa is T
   Ordinate is Electrical_Resistance
   Begin Values
      # [K]      [Ohm-um]
      273        1.00E-9
      323        7.99E-10
      ...
      873        1.09E-11
   End Values
End definition for function RESISTANCE_DATA

...

Begin Aria Material Foo
   ...
   Electrical Resistance = User_Function Name=RESISTANCE_DATA X=Temperature
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the resistance as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`RESISTANCE_DATA` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.12 EMISSIVITY

EMISSIVITY = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description

Specifies the material model for the emissivity.

Summary

Specifies the material model for the emissivity.

Parent Block(s) `ARIA MATERIAL`

4.12.1 EMISSIVITY = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example
```
Emissivity = Calore_User_Sub name=w80afftuser type=element NR=2 R0=1000
R1=0.7
```

Description `NAME` is the name of the user subroutine was registered with. `NR` and `NI` are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the `R`n (`I`n) parameter. **Note:** the parameters use zero based counting. The `TYPE` parameter denotes the API the user subroutine uses; currently Aria only supports types `ELEMENT` and `NODE`.

The optional `MULTIPLIER` parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

4.12.2 EMISSIVITY = CONSTANT

Parameters
```
E = REAL
```

Example
```
Emissivity = Constant E = 0.8
```

Description `E` is the value of the constant emissivity.

4.12.3 EMISSIVITY = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Emissivity= Polynomial Variable=Temperature Order=1 C0=401.0 C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$\epsilon = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.32}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, X_o is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.12.4 EMISSIVITY = ENCORE_FUNCTION

Parameters

```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example

```
Emissivity = Encore_Function Name=My_Function
```

Description

Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.12.5 EMISSIVITY = USER_FUNCTION

Parameters

```
NAME = STRING
X = STRING
```

Example

```
Begin Definition for Function My_Emissivity
   Type is Piecewise Linear
   Begin Values
     100 0.7
     200 0.6
     300 0.5
   End
End
...

Begin Aria Material Foo
   ...
   Emissivity = User_Function Name=My_Emissivity X=Temperature
   ...
End Aria Material Foo
```

Description A look-up function is used to compute the values of the emissivity as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`My_Emissivity` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.13 ENTHALPY

ENTHALPY = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies a model for the enthalpy of a material.

Summary Specifies a model for the enthalpy of a material.

Parent Block(s) `ARIA MATERIAL`

4.13.1 ENTHALPY = CONSTANT

Parameters
```
H = REAL
```

Example
```
Enthalpy = Constant H=1e-4
```

Description The value is constant in space and time.

4.13.2 ENTHALPY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Enthalpy = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.14 EQUATION OF STATE

EQUATION OF STATE = ***MODEL*** [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the equation of state for gas dynamics problems.

Summary Specifies the equation of state for gas dynamics problems.

Parent Block(s) `ARIA MATERIAL`

4.14.1 EQUATION OF STATE = IDEAL_GAS

Parameters
```
R = REAL
GAMMA = REAL
```

Example
```
Equation of State = Ideal_Gas R=8.314 Gamma=1.4
```

Description `R` is the gas constant and `GAMMA` is the ratio of heat capacities.

This model is used for gas dynamics problems where the density is an unknown. In this case, the pressure is given by the ideal gas law,

$$p = RT\rho \tag{4.33}$$

where T is the temperature and ρ is the density. Activating this model supplies several quantities that are related to this equation of state such as the pressure, temperature, and other gas dynamics related quantities.

4.15 HEAT CONDUCTION

HEAT CONDUCTION = ***MODEL*** [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material (constitutive) model for the heat conduction (diffusive flux) in the bulk.

Summary Specifies the material (constitutive) model for the heat conduction (diffusive flux) in the bulk.

Parent Block(s) `ARIA MATERIAL`

4.15.1 HEAT CONDUCTION = BASIC

This is an alias for `FOURIERS_LAW`.

Example `Heat Conduction = Basic`

4.15.2 HEAT CONDUCTION = CONVECTED_ENTHALPY

Parameters `(none)`

Example `Heat Conduction = Convected_Enthalpy`

Description The heat conduction (flux) $\boldsymbol{q}$ is given by,

$$\boldsymbol{q} = -h\rho\boldsymbol{v} \tag{4.34}$$

where h is the enthalpy, ρ is the density and $\boldsymbol{v}$ is the velocity.

4.15.3 HEAT CONDUCTION = FOURIERS_LAW

Parameters `(none)`

Example `Heat Conduction = Fouriers_Law`

Description The heat conduction (flux) $\boldsymbol{q}$ is given by Fourier's Law,

$$\boldsymbol{q} = -\kappa\boldsymbol{\nabla}T \tag{4.35}$$

where κ is the thermal conductivity and T is the temperature.

4.15.4 HEAT CONDUCTION = GENERALIZED

Parameters `(none)`

Example `Heat Conduction = Generalized`

Description The heat conduction (flux) $\boldsymbol{q}$ is given by Fourier's Law,

$$\boldsymbol{q} = -\kappa\boldsymbol{\nabla}T \tag{4.36}$$

where κ is the thermal conductivity tensor and T is the temperature.

4.15.5 HEAT CONDUCTION = THERMOELECTRIC

Parameters `(none)`

Example
```
Heat Conduction = Thermoelectric
```

Description The heat conduction (flux) $\boldsymbol{q}$ is given by Fourier's Law combined with an electric energy flux,

$$\boldsymbol{q} = -\kappa \boldsymbol{\nabla} T + \alpha T J \tag{4.37}$$

where κ is the thermal conductivity, α is the Seebeck coefficient, T is the temperature and J is the thermoelectric current density 4.6.4.

4.16 HEAT OF VAPORIZATION

HEAT OF VAPORIZATION = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the heat of vaporization for a material or for a particular species.

Summary Quantifies the amount of energy consumed during evaporation per unit mass.

Parent Block(s) `ARIA MATERIAL`

4.16.1 HEAT OF VAPORIZATION = CONSTANT

Parameters
```
Hv = REAL
[SUBINDEX = INT]
```

Example
```
HEAT OF VAPORIZATION = CONSTANT SUBINDEX=0 Hv = 1.0
HEAT OF VAPORIZATION = CONSTANT SUBINDEX=1 Hv = 2.0
HEAT OF VAPORIZATION = CONSTANT SUBINDEX=3 Hv = 3.14
```

Description `HV` is the value of the constant heat of vaporization and `SUBINDEX` is the optional species index (used in multicomponent systems).

4.17 HEAT TRANSFER COEFFICIENT

HEAT TRANSFER COEFFICIENT = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the heat transfer coefficient.

Summary Specifies the material model for the heat transfer coefficient.

Parent Block(s) `ARIA MATERIAL`

4.17.1 HEAT TRANSFER COEFFICIENT = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example
```
Heat Transfer Coefficient = Calore_User_Sub name=w80afftuser type=element
NR=2 R0=1000 R1=0.7
```

Description

NAME is the name of the user subroutine was registered with. NR and NI are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the Rn (In) parameter. **Note:** the parameters use zero based counting. The TYPE parameter denotes the API the user subroutine uses; currently Aria only supports types ELEMENT and NODE.

The optional MULTIPLIER parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

4.17.2 HEAT TRANSFER COEFFICIENT = CONSTANT

Parameters
```
H = REAL
```

Example
```
Heat Transfer Coefficient = Constant h = 0.8
```

Description

H is the value of the constant heat transfer coefficient.

4.17.3 HEAT TRANSFER COEFFICIENT = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Heat Transfer Coefficient = Encore_Function Name=My_Function
```

Description

Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.17.4 HEAT TRANSFER COEFFICIENT = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Heat Transfer Coefficient= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$h = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.38}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.17.5 HEAT TRANSFER COEFFICIENT = USER_FUNCTION

Parameters

```
NAME = STRING
X = STRING
```

Example

```
Begin Definition for Function My_Heat_Transfer_Coefficient
   Type is Piecewise Linear
   Begin Values
     100 0.7
     200 0.6
     300 0.5
   End
End
...

Begin Aria Material Foo
   ...
   Heat Transfer Coefficient = User_Function Name=My_Heat_Transfer_Coefficient
         X=Temperature
   ...
End Aria Material Foo
```

Description A look-up function is used to compute the values of the heat transfer coefficient as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`My_Heat_Transfer_Coefficient` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.18 INTRINSIC PERMEABILITY

INTRINSIC PERMEABILITY = *MODEL* [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the intrinsic permeability tensor for porous flow in Darcy's Law.

Summary Specifies the material model for the intrinsic permeability tensor for porous flow in Darcy's Law. In general, the permeability may be nonisotropic in porous media. In that case, Darcy's law may be written as,

$$\rho \boldsymbol{v}_d = \boldsymbol{f} = -\frac{\rho k_r}{\mu} \boldsymbol{K} \cdot (\boldsymbol{\nabla} P - \rho \boldsymbol{g}) \tag{4.39}$$

where $\boldsymbol{K}$ is the intrinsic permeability tensor, ρ is the density, $\boldsymbol{v}_d$ is the Darcy velocity, $\boldsymbol{f}$ is the mas flux, k_r is the relative permeability, P is pressure and $\boldsymbol{g}$ is gravity.

Parent Block(s) `ARIA MATERIAL`

4.18.1 INTRINSIC PERMEABILITY = CONSTANT

Parameters
```
[XX = REAL]
[XY = REAL]
[XZ = REAL]
[YX = REAL]
[YY = REAL]
[YZ = REAL]
[ZX = REAL]
[ZY = REAL]
[ZZ = REAL]
```

Example
```
Intrinsic Permeability = Constant XX=1 YY=2 ZZ=1
```

Description All components default to zero and all values are constant in space and time.

4.19 INTERNAL ENERGY

INTERNAL ENERGY = *MODEL* [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies a model for the internal energy of a material.

Summary Specifies a model for the internal of a material.

Parent Block(s) `ARIA MATERIAL`

4.19.1 INTERNAL ENERGY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Internal Energy = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.19.2 INTERNAL ENERGY = GAS_PHASE

Parameters `(none)`

Example
```
Internal Energy = Gas_Phase
```

Description The internal energy e is computed using thermodynamic relation

$$e = h - P/\rho \tag{4.40}$$

where h is the enthalpy, P is the (partial) pressure and ρ is the density.

4.20 LEVEL SET HEAVISIDE

LEVEL SET HEAVISIDE = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the functional form of the Heaviside function used with level set algorithms. This also implies the Dirac delta function used for level set algorithms.

Summary Specifies the functional form of the Heaviside function used with level set algorithms. This also implies the Dirac delta function used for level set algorithms.

Parent Block(s) `ARIA MATERIAL`

4.20.1 LEVEL SET HEAVISIDE = SMOOTH

Parameters `(none)`

Example `LEVEL SET HEAVISIDE = SMOOTH`

Description The Heaviside function in this case is given as

$$H(f) = \frac{1}{2}\left[1 + f/w + \sin\left(\pi f/w\right)/\pi\right] \tag{4.41}$$

Here f is the level set distance function and w is half of the level set width (see 4.21).

4.21 LEVEL SET WIDTH

LEVEL SET WIDTH = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the total width of the level set interface. Half of this width falls on the positive side of the zero level set and half falls on the negative side.

Summary Specifies the total width of the level set interface. Half of this width falls on the positive side of the zero level set and half falls on the negative side.

Parent Block(s) `ARIA MATERIAL`

4.21.1 LEVEL SET WIDTH = CONSTANT

Parameters `WIDTH = REAL`

Example `LEVEL SET WIDTH = CONSTANT WIDTH=0.1`

Description This is what you'd expect it to be – a uniform constant everywhere for all time.

4.22 LUBRICATION HEIGHT LOWER

LUBRICATION HEIGHT LOWER = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the lower height function for the lubrication equation.

Summary Specifies the material model for the lower height function for the lubrication equation.

Parent Block(s) `ARIA MATERIAL`

4.22.1 LUBRICATION HEIGHT LOWER = CONSTANT

Parameters `H_LOWER = REAL`

Example `LUBRICATION HEIGHT LOWER = CONSTANT H_LOWER = 1.0`

Description `H_LOWER` is the constant value.

4.23 LUBRICATION HEIGHT UPPER

LUBRICATION HEIGHT UPPER = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the upper height function for the lubrication equation.

Summary Specifies the material model for the upper height function for the lubrication equation.

Parent Block(s) `ARIA MATERIAL`

4.23.1 LUBRICATION HEIGHT UPPER = CONSTANT

Parameters `H_UPPER = REAL`

Example `LUBRICATION HEIGHT UPPER = CONSTANT H_UPPER = 1.0`

Description `H_UPPER` is the constant value.

4.24 LUBRICATION K

LUBRICATION K = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the "k" model for the lubrication equation.

Summary In the standard derivation of Reynolds' lubrication theory, the flow is assumed to be laminar and the constant "12" appears in the equation. However, there are extensions to the standard lubrication theory that allow studying turbulent flow by replacing the "12" with a model for turbulent mixing, a scalar that we call K.

If this line is omitted, the value of $K = 12$ is assumed. However, other forms for this constant K may be used.

Parent Block(s) `ARIA MATERIAL`

4.24.1 LUBRICATION K = CONSTANT

Parameters `K = REAL`

Example `LUBRICATION K = CONSTANT K = 12.0`

Description `K` is the constant value. Defaults to $K = 12$ if this line is omitted.

4.24.2 LUBRICATION K = PRANDTL_MIXING

Parameters `(none)`

Example `LUBRICATION K = PRANDTL_MIXING`

Description This model calculates the turbulent mixing viscosity based on the standard Prandtl mixing theory. First, the Reynolds' number is calculated using

$$\mathrm{Re} = \frac{h\rho\sqrt{\mathrm{V_{UPPER}} - \mathrm{V_{LOWER}}}}{\mu}. \tag{4.42}$$

Then, the value of K is calculated with

$$K = \begin{cases} 12 & \mathrm{Re} < 2000 \\ 0.03(2\mathrm{Re})^{0.75} & \mathrm{Re} > 2000 \end{cases}. \tag{4.43}$$

4.25 LUBRICATION VELOCITY LOWER

LUBRICATION VELOCITY LOWER = *MODEL* [$\mathrm{param}_1 = \mathrm{val}_1$, $\mathrm{param}_2 = \mathrm{val}_2 \ldots$]

Description Specifies the material model for the velocity of the lower boundary for the lubrication equation.

Summary Specifies the material model for the velocity of the lower boundary for the lubrication equation.

Parent Block(s) `ARIA MATERIAL`

4.25.1 LUBRICATION VELOCITY LOWER = CONSTANT

Parameters `V_LOWER = REAL`

Example `LUBRICATION VELOCITY LOWER = CONSTANT V_LOWER = 0.0`

Description `V_LOWER` is the constant value.

4.26 LUBRICATION VELOCITY UPPER

LUBRICATION VELOCITY UPPER = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the velocity of the upper boundary for the lubrication equation.

Summary Specifies the material model for the velocity of the upper boundary for the lubrication equation.

Parent Block(s) `ARIA MATERIAL`

4.26.1 LUBRICATION VELOCITY UPPER = CONSTANT

Parameters `V_UPPER = REAL`

Example `LUBRICATION VELOCITY UPPER = CONSTANT V_UPPER = 0.0`

Description `V_UPPER` is the constant value.

4.27 MASS FLUX

MASS FLUX = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies a constitutive model for the mass flux for porous flow applications.

Summary Specifies a constitutive model for the mass flux for porous flow applications.

Parent Block(s) `ARIA MATERIAL`

4.27.1 MASS FLUX = DARCY

Parameters
```
[GX = REAL]
[GY = REAL]
[GZ = REAL]
```

Example
```
Mass Flux = Darcy GY=-9.8
```

Description The macroscopic, convective mass flux in phase β, $\rho\boldsymbol{v}$ is obtained from the extended Darcy's Law,

$$\boldsymbol{F} = \rho\boldsymbol{f} = -\frac{\rho k_r}{\mu}\boldsymbol{K}\cdot(\boldsymbol{\nabla}P - \rho\boldsymbol{g}) \tag{4.44}$$

4.28 MESH LAMBDA

MESH LAMBDA = *MODEL* [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the λ Lamé coefficient for the mesh (psuedo-solid) stress tensor.

Summary The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk}\boldsymbol{I} + 2\mu\boldsymbol{E} - \beta\left(T - T_{ref}\right)\boldsymbol{I} \tag{4.45}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.46}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu\frac{\nu}{(1-2\nu)} \tag{4.47}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha(3\lambda + 2\mu) \tag{4.48}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Parent Block(s) `ARIA MATERIAL`

4.28.1 MESH LAMBDA = CONSTANT

Parameters `L = REAL`

Example `MESH LAMBDA = CONSTANT L = 1.0`

Description `L` is the value of the constant λ.

4.28.2 MESH LAMBDA = CONVERTED

Parameters `(None)`

Example `MESH LAMBDA = Converted`

Description Aria will use Young's modulus and Poisson ratio to compute the Lamé λ coefficient. Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

4.28.3 MESH LAMBDA = ENCORE_FUNCTION

Parameters `NAME = STRING`
`[EVAL_TYPE = STRING]`

Example `MESH LAMBDA = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.29 MESH TWO MU

MESH TWO MU = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for twice the μ Lamé coefficient for the mesh (psuedo-solid) stress tensor.

Summary The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk} \boldsymbol{I} + 2\mu \boldsymbol{E} - \beta \left(T - T_{ref}\right) \boldsymbol{I} \tag{4.49}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.50}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu \frac{\nu}{(1-2\nu)} \tag{4.51}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha (3\lambda + 2\mu) \tag{4.52}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Parent Block(s) `ARIA MATERIAL`

4.29.1 MESH TWO MU = CONSTANT

Parameters `TWO_MU = REAL`

Example `MESH TWO MU = CONSTANT TWO_MU = 1.0`

Description `TWO_MU` is the value of 2μ.

4.29.2 MESH TWO MU = CONVERTED

Parameters `(None)`

Example `MESH TWO MU = Converted`

Description Aria will use Young's modulus and Poisson ratio to compute the Lamé μ coefficient. Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

4.29.3 MESH TWO MU = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
TWO_MU = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.30 MESH STRESS

MESH STRESS = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies a contribution to the mesh (pseudo-solid) stress tensor. Multiple stresses are combined additively and may be specified by using this line command multiple times.

Summary Specifies a contribution to the mesh (pseudo-solid) stress tensor. The total stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \sum_j \boldsymbol{T}_j \tag{4.53}$$

Parent Block(s) `ARIA MATERIAL`

4.30.1 MESH STRESS = ISOTHERMAL

Parameters
```
T = REAL
T_REF = REAL
```

Example
```
MESH STRESS = Isothermal T=500 T_ref=325
```

Description This stress accounts for the mechanical stresses due to thermally induced strains.

$$\boldsymbol{T} = -\beta\,(T - T_{ref})\,\boldsymbol{I} \tag{4.54}$$

where β is the Lamé coefficient of thermal stress (related to the coefficient of thermal expansion, α), T is the temperature and T_{ref} is the temperature of the undeformed reference state of the mesh (pseudo-solid). This is a specialization of the `THERMAL` model that uses uniform, fixed temperature and thermal strain reference temperature.

4.30.2 MESH STRESS = LINEAR_ELASTIC

Parameters `REFERENCE_FRAME = "MOVING" | "UNDEFORMED"`

Example `MESH STRESS = Linear_Elastic Reference_Frame = undeformed`

Description Supplies the linear elasticity stress tensor,

$$\boldsymbol{T} = \lambda\,\mathrm{trace}\,\boldsymbol{E}\boldsymbol{I} + 2\mu\boldsymbol{E} \tag{4.55}$$

where λ and μ and the Lamé coefficients and $\boldsymbol{E}$ is the strain tensor. When the choice of reference frame is "UNDEFORMED" then the strain is computed in the undeformed reference state; this is commonly referred to as small strain theory. When the reference frame is "MOVING" then the strain is computed with respect to the deformed coordinates.

Specifically, the strain tensor is given by

$$\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^t\right) \tag{4.56}$$

where $\boldsymbol{d}$ is the mesh displacement field. The choice of reference frame determines whether the $\boldsymbol{\nabla}$ operator is computed in the undeformed or moving reference frames.

4.30.3 MESH STRESS = NEOHOOKEAN_ELASTIC

Parameters `(none)`

Example `MESH STRESS = Neohookean_Elastic`

Description Supplies a nonlinear hyperelastic stress of the form,

$$\boldsymbol{T} = \frac{\mu}{J}\left(\boldsymbol{b} - \boldsymbol{I}\right) + \frac{\lambda}{J}\ln J\boldsymbol{I} \tag{4.57}$$

where λ and μ and the Lamé coefficients, $\boldsymbol{b} \equiv \boldsymbol{F}\cdot\boldsymbol{F}^t$ is the left Cauchy-Green tensor, $\boldsymbol{F}$ is the deformation gradient and $J \equiv \det \boldsymbol{F}$. See, e.g., [4] or [5].

4.30.4 MESH STRESS = NONLINEAR_ELASTIC

Parameters `(none)`

Example `MESH STRESS = Nonlinear_Elastic`

Description Supplies a nonlinear elastic stress,

$$\boldsymbol{T} = \lambda \operatorname{trace} \boldsymbol{E} \boldsymbol{I} + 2\mu \boldsymbol{E} \tag{4.58}$$

where λ and μ and the Lamé coefficients and $\boldsymbol{E}$ is the strain tensor. The particular choice of strain tensor chosen depends on where the configuration (reference frame) which is set via the MESH MOTION command line. See section 5.16 for more information. When the MESH MOTION is set to REFERENCE_FRAME then the Green strain is used. Otherwise, the Almansi strain is used.

4.30.5 MESH STRESS = RESIDUAL

Parameters
```
[SXX | SX = REAL]
[SYY | SY = REAL]
[SZZ | SZ = REAL]
[SXY = REAL]
[SXZ = REAL]
[SYZ = REAL]
```

Example `SOLID STRESS = Residual Sxx=0.02 Syy=0.02`

Description This stress accounts for the initial residual stress in a solid that is constant and uniform everywhere. The components of the residual stress tensor are supplied by the (up-to) six components `SXX`, `SYY`, `SZZ`, `SXY`, `SXZ`, and `SYZ`.

This is directly analogous to the `ISTRESS` condition in ANSYS. To that end, the diagonal components can be specified as either `SXX` or SX etc.

4.30.6 MESH STRESS = THERMAL

Parameters `(none)`

Example `MESH STRESS = Thermal`

Description This stress accounts for the mechanical stresses due to thermally induced strains.

$$\boldsymbol{T} = -\beta \left(T - T_{ref}\right) \boldsymbol{I} \tag{4.59}$$

where β is the Lamé coefficient of thermal stress (related to the coefficient of thermal expansion, α), T is the temperature and T_{ref} is the temperature of the undeformed reference state of the mesh (pseudo-solid).

4.30.7 MESH STRESS = LAME

Parameters `MODEL = STRING`

Example `MESH STRESS = Lame Model = Neo_Hookean`

Description This stress interfaces with the Lame constitutive model library used by Sierra/SolidMechanics. As such, it allows the use of many of Lame's constitutive models. Sierra/Multimechanics prepares the kinematic quantities, and Lame returns the Cauchy stress, which is integrated in the `MESH` equation.

Material parameters for the constitutive models are handled by an input block consistent with the Sierra/SolidMechanics formatting. For documentation of the available constitutive models and the required parameter input, please see the Sierra/SolidMechanics documentation.

Currently this interface is instrumented to support hyperelastic constitutive models, and is restricted to the following models:

- `NEO_HOOKEAN`
- `THERMALBATTERYSEPARATOR`
- `SOILFOAMPHASETRANSITION`

4.31 MOLECULAR WEIGHT

MOLECULAR WEIGHT = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the molecular weight for a species.

Summary Specifies the molecular weight for a species.

Parent Block(s) `ARIA MATERIAL`

4.31.1 MOLECULAR WEIGHT = CONSTANT

Parameters
```
M = REAL
[SUBINDEX = INT]
```

Example
```
Molecular Weight = Constant Subindex=1 M = 17.0
Molecular Weight = Constant Subindex=2 M = 23.0
Molecular Weight = Constant Subindex=5 M = 34.0
```

Description `M` is the value of the molecular weight.

`SUBINDEX` is the species subindex.

4.31.2 MOLECULAR WEIGHT = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example `Molecular Weight = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.32 MOMENTUM STRESS

MOMENTUM STRESS = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies a contribution to the fluid stress tensor. Multiple stresses are combined additively and may be specified by using this line command multiple times.

Summary Specifies a contribution to the fluid momentum stress tensor. The total stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \sum_j \boldsymbol{T}_j \tag{4.60}$$

Parent Block(s) `ARIA MATERIAL`

4.32.1 MOMENTUM STRESS = LS_CAPILLARY

Parameters `(none)`

Example
```
MOMENTUM STRESS = Newtonian
MOMENTUM STRESS = LS_Capillary
```

Description This adds the capillary boundary condition contributions in the vicinity of the level set interface.

$$\boldsymbol{T} = \sigma\delta(F)\left(\boldsymbol{I} - \boldsymbol{N}\boldsymbol{N}\right) \tag{4.61}$$

where σ is the surface tension, $\delta(F)$ is the level set delta function, F is the level set distance function and $\boldsymbol{N}$ is the level set normal field.

4.32.2 MOMENTUM STRESS = INCOMPRESSIBLE_NEWTONIAN

Parameters `[PRESSURE = STRING]`

Example `MOMENTUM STRESS = Incompressible_Newtonian`

Description Supplies the incompressible Newtonian stress tensor,

$$\boldsymbol{T} = -p\boldsymbol{I} + \mu\left(\boldsymbol{\nabla v} + \boldsymbol{\nabla v}^t\right) \tag{4.62}$$

where μ is the fluid viscosity, p is the pressure and $\boldsymbol{v}$ is the fluid velocity. The viscosity μ is provided with the viscosity line command as described in section 4.56.

Note that this model does not include stress contributions that are proportional to the divergence of the velocity. To incorporate those contributions, use the NEWTONIAN_DILATIONAL stress model in addition to this model, or use the FORMAL_NEWTONIAN stress model instead of this model.

By default, the pressure field used by this model is PRESSURE with the same subindex as the model. The optional parameter `PRESSURE` can be used to specify an alternate pressure field. This is useful for running problems with a continuous velocity field but a discontinuous pressure field.

4.32.3 MOMENTUM STRESS = FORMAL_NEWTONIAN

Parameters `(none)`

Example `MOMENTUM STRESS = Formal_Newtonian`

Description Supplies the complete Newtonian stress tensor,

$$\boldsymbol{T} = -p\boldsymbol{I} + \mu\left(\boldsymbol{\nabla v} + \boldsymbol{\nabla v}^t\right) + \left(\kappa - \frac{2}{3}\mu\right)\boldsymbol{\nabla \cdot v}\boldsymbol{I} \tag{4.63}$$

where μ is the fluid viscosity, p is the pressure and $\boldsymbol{v}$ is the fluid velocity. The viscosity μ is provided with the viscosity line command as described in section 4.56. The bulk viscosity κ is provided with the bulk viscosity line command as described in section 4.4.

4.32.4 MOMENTUM STRESS = NEWTONIAN_DILATIONAL

Parameters `(none)`

Example `MOMENTUM STRESS = Newtonian_Dilational`

Description Adds the dilational stress contribution for Newtonian fluids,

$$\boldsymbol{T} = \left(\kappa - \frac{2}{3}\mu\right)\boldsymbol{\nabla \cdot v}\boldsymbol{I} \tag{4.64}$$

where κ is the bulk viscosity of the fluid, μ is the fluid (dynamic) viscosity and $\boldsymbol{v}$ is the fluid velocity. The viscosity μ is provided with the viscosity line command as described in section 4.56. The bulk viscosity κ is provided with the bulk viscosity line command as described in section 4.4.

See, also, the NEWTONIAN momentum stress model.

4.32.5 MOMENTUM STRESS = NEWTONIAN_VISCOUS

Parameters (none)

Example `MOMENTUM STRESS = Newtonian_Viscous`

Description Supplies only the viscous contribution of the Newtonian stress tensor,

$$\boldsymbol{T} = \mu \left(\boldsymbol{\nabla v} + \boldsymbol{\nabla v}^t \right) \tag{4.65}$$

where μ is the fluid viscosity and $\boldsymbol{v}$ is the fluid velocity. The viscosity μ is provided with the viscosity line command as described in section 4.56.

4.32.6 MOMENTUM STRESS = NEWTONIAN_PRESSURE

Parameters (none)

Example `MOMENTUM STRESS = Newtonian_Pressure`

Description Supplies only the pressure contribution of the Newtonian stress tensor,

$$\boldsymbol{T} = -p\boldsymbol{I} \tag{4.66}$$

where p is the pressure.

4.32.7 MOMENTUM STRESS = MAXWELL

Parameters (none)

Example `MOMENTUM STRESS = Maxwell`

Description Includes the force due to an electric field (neglecting magnetic effects) due to the Maxwell stress tensor,

$$\boldsymbol{T}_M = \varepsilon \left(\mathbf{E}\mathbf{E}^T - \frac{1}{2}(\mathbf{E} \cdot \mathbf{E})\boldsymbol{I} \right) \tag{4.67}$$

where ε is the electrical permittivity and $\mathbf{E}$ is the electric field. The divergence of the Maxwell stress tensor is a force applied to the momentum equation,

$$\boldsymbol{\nabla} \cdot \boldsymbol{T}_M = \rho_e \mathbf{E} - \frac{1}{2}(\mathbf{E} \cdot \mathbf{E})\boldsymbol{\nabla}\varepsilon, \tag{4.68}$$

where ρ_e is the volumetric charge density.

4.33 POISSONS RATIO

POISSONS RATIO = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the Poisson's ratio.

Summary The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk}\boldsymbol{I} + 2\mu\boldsymbol{E} - \beta\left(T - T_{ref}\right)\boldsymbol{I} \tag{4.69}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.70}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu\frac{\nu}{(1-2\nu)} \tag{4.71}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha\left(3\lambda + 2\mu\right) \tag{4.72}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

Parent Block(s) `ARIA MATERIAL`

4.33.1 POISSONS RATIO = CONSTANT

Parameters
```
PR = REAL
```

Example
```
POISSONS RATIO = CONSTANT PR = 1.0
```

Description `PR` is the value of the constant Poisson's ratio.

4.33.2 POISSONS RATIO = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
POISSONS RATIO = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.34 POROSITY

POROSITY = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the porosity for porous flow applications.

Summary Specifies the material model for the porosity for porous flow applications.

Parent Block(s) `ARIA MATERIAL`

4.34.1 POROSITY = CONSTANT

Parameters `PHI = REAL`

Example `Porosity = Constant PHI=0.1`

Description The value is constant in space and time.

4.34.2 POROSITY = DEFORMING

Parameters `PHI_INIT = REAL`

Example `Porosity = Deforming`

Description The porosity evolves from an initial value in accordance with a deforming media.

$$\phi = 1 - (1 - \phi_\circ)\, e^{\epsilon_v} \tag{4.73}$$

where ϵ_v is the volume strain,

$$\epsilon_v = \boldsymbol{\nabla} \cdot \boldsymbol{d} \tag{4.74}$$

where $\boldsymbol{d}$ is the total mesh displacement field. Here $\phi_\circ$ is the initial porosity value given by the PHI_INIT parameter.

4.34.3 POROSITY = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Porosity= Polynomial Variable=Temperature Order=1 C0=401.0 C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$\phi = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.75}$$

Here, N is the order of the polynomial provided by the ORDER parameter and X is the variable supplied by the VARIABLE parameter, $X_\circ$ is an optional offset (VARIABLE_OFFSET, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The VARIABLE argument can be TIME or any internal Expression that evaluates to a scalar. For the latter case, the format of the VARIABLE argument is described in section 2.7.1.

4.34.4 POROSITY = ROCK_COMPRESSIBLE

Parameters

```
REF_POROSITY = REAL
REF_PRESSURE = REAL
CR = REAL
```

Example

```
Porosity = Rock_Compressible Ref_Porosity=0.1 Ref_Pressure=101325 Cr=-1e-7
```

Description The compressibility of a porous material is defined as

$$C_r = \frac{1}{\phi}\frac{\partial \phi}{\partial P} \tag{4.76}$$

where ϕ is the porosity and P is pressure. Using this definition, we define an approximate model for the porosity as a function of pressure,

$$\phi = \phi_{ref}\left(1 + C_r\left(P - P_{ref}\right)\right) \tag{4.77}$$

where ϕ_{ref} is given by the REF_POROSITY parameter and is the porosity at the reference pressure P_{ref} which is given by the REF_PRESSURE parameter. The compressibility C_r is given by the CR parameter.

4.35 RADIATION FORM FACTOR

RADIATION FORM FACTOR = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the radiative heat transfer form factor used in the GENERALIZED_RAD boundary condition.

Summary The form factor represents the fraction of the radiation leaving a surface which interacts with a body at another temperature. Thus the form factor should be less than or equal to unity.

Parent Block(s) `ARIA MATERIAL`

4.35.1 RADIATION FORM FACTOR = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example
```
Radiation Form Factor = Calore_User_Sub name=w80afftuser type=element NR=2
R0=1000 R1=0.7
```

Description `NAME` is the name of the user subroutine was registered with. `NR` and `NI` are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the `R`n (`I`n) parameter. **Note:** the parameters use zero based counting. The `TYPE` parameter denotes the API the user subroutine uses; currently Aria only supports types `ELEMENT` and `NODE`.

The optional `MULTIPLIER` parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

4.35.2 RADIATION FORM FACTOR = CONSTANT

Parameters
```
F = REAL
```

Example
```
Radiation Form Factor = Constant F = 0.8
```

Description `F` is the value of the constant radiation form factor.

4.35.3 RADIATION FORM FACTOR = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Radiation Form Factor= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$F = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.78}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.35.4 RADIATION FORM FACTOR = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Radiation Form Factor = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.35.5 RADIATION FORM FACTOR = USER_FUNCTION

Parameters
```
NAME = STRING
X = STRING
```

Example

```
Begin Definition for Function My_Radiation_Form_Factor
   Type is Piecewise Linear
   Begin Values
     100 0.7
     200 0.6
     300 0.5
   End
End
...

Begin Aria Material Foo
   ...
   Radiation Form Factor = User_Function Name=My_Radiation_Form_Factor X=Temperature
   ...
End Aria Material Foo
```

Description A look-up function is used to compute the values of the form factor as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`My_Radiation_Form_Factor` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.36 RADIATIVE CONDUCTIVITY

RADIATIVE CONDUCTIVITY = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the radiative effective thermal conductivity.

Summary Specifies the material model for the radiative portion of thermal conductivity that appears in the diffusion term of the energy equation for temperature.

Parent Block(s) `ARIA MATERIAL`

4.36.1 RADIATIVE CONDUCTIVITY = CONSTANT

Parameters `Krad = REAL`

Example

```
RADIATIVE CONDUCTIVITY = CONSTANT Krad = 1.0
```

Description `Krad` is the value of the constant radiative thermal conductivity.

4.36.2 RADIATIVE CONDUCTIVITY = CHEMEQ_FOAM

Parameters
```
Kc0 = REAL
Kc1 = REAL
```

Example
```
Radiative Conductivity = ChemEq_Foam Kc0 = 0.00268 Kc1 = 0.0
```

Description For a reacting foam defined in ChemEq, this model uses

$$k_{rad} = \frac{Kc_0}{Kc_1 + \sum_s Y_i} \sigma T^3 \tag{4.79}$$

where the denominator includes the sum of the mass fractions of the solid species of the foam. This uses the species phase definition from the ChemEq block to determine which species are solid. This model can only be used in a material block which also contains a ChemEq definition.

This uses expressions for the mass fractions, so its material block must also include "Mass_Fraction of X = From_ChemEq" for each solid species.

The `GLOBAL CONSTANTS` block must include include the `STEFAN-BOLTZMANN CONSTANT` entry to provide a value of σ.

4.36.3 RADIATIVE CONDUCTIVITY = OPTICALLY_THICK

Parameters
```
K = REAL
N = REAL
BETA_R = REAL
```

Example
```
RADIATIVE CONDUCTIVITY = OPTICALLY_THICK N = 2.1 BETA_R = 1.0e-5
```

Description Thermal conductivity used to model effective radiative conductivity diffusion in the optically-thick limit. Constant values for index of refraction n, and Rosseland-mean extinction coefficient β_R must be provided on the input line.

$$k_{rad} = \frac{16\sigma n^2}{3\beta_R} T^3 \tag{4.80}$$

The `GLOBAL CONSTANTS` block must include include the `STEFAN-BOLTZMANN CONSTANT` entry to provide a value of σ.

4.36.4 RADIATIVE CONDUCTIVITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Radiative Conductivity = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument EVAL_TYPE can be used to select the particular function on the Encore function object. Valid options include VALUE, DOT, GRADIENT, FLUX and STRESS. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – VALUE for NO_OP expressions, DOT for DT_OP expressions and GRADIENT for GRAD_OP expressions.

4.36.5 RADIATIVE CONDUCTIVITY = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Radiative Conductivity= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$\kappa = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.81}$$

Here, N is the order of the polynomial provided by the ORDER parameter and X is the variable supplied by the VARIABLE parameter, $X_\circ$ is an optional offset (VARIABLE_OFFSET, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The VARIABLE argument can be TIME or any internal Expression that evaluates to a scalar. For the latter case, the format of the VARIABLE argument is described in section 2.7.1.

4.36.6 RADIATIVE CONDUCTIVITY = USER_FUNCTION

Parameters

```
NAME = STRING
X = STRING
```

Example

```
begin definition for function SI_Kr
  type is piecewise linear
  begin values
     20.0     5.50e7
    100.0     4.60e7
    ...
    800.0     1.30e7
   2000.0     1.30e7
  end values
end definition for function SI_Kr

...

Begin Aria Material Foo
   ...
   Radiative Conductivity = User_Function Name=SI_Kr X=Temperature
   ...
End Aria Material Foo
```

Description A look-up function is used to compute the values of the thermal conductivity as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function for (`CONDUCTIVITY_DATA`, SI_K, in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the optional abscissa variable identified in the user-defined function.

4.37 RELATIVE PERMEABILITY

RELATIVE PERMEABILITY = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the relative permeability (scalar) for porous flow in Darcy's Law.

Summary Specifies the material model for the relative permeability (scalar) for porous flow in Darcy's Law. In general, the permeability may be nonisotropic in porous media. In that case, Darcy's law may be written as,

$$\rho \boldsymbol{v}_d = \boldsymbol{f} = \frac{\rho k_r}{\mu} \boldsymbol{K} \cdot (\boldsymbol{\nabla} P + \rho \boldsymbol{g}) \tag{4.82}$$

where $\boldsymbol{K}$ is the intrinsic permeability tensor, ρ is the density, $\boldsymbol{v}_d$ is the Darcy velocity, $\boldsymbol{f}$ is the mass flux, k_r is the relative permeability, μ is the dynamic viscosity, P is pressure and $\boldsymbol{g}$ is gravity.

Parent Block(s) `ARIA MATERIAL`

4.37.1 RELATIVE PERMEABILITY = CONSTANT

Parameters `K = REAL`

Example `Relative Permeability = Constant K=1e-3`

Description The value is constant in space and time.

4.37.2 RELATIVE PERMEABILITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example `Relative Permeability = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.37.3 RELATIVE PERMEABILITY = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Relative Permeability= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$k_r = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.83}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.38 SEEBECK COEFFICIENT

SEEBECK COEFFICIENT = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for Seebeck coefficient.

Summary Specifies the material model for the Seebeck coefficient that appears in a thermoelectric contribution to the the diffusion term of the energy equation for temperature. This command line must be accompanied the `HEAT_CONDUCTION = THERMOELECTRIC` 4.15.5 and `CURRENT_DENSITY = THERMOELECTRIC` 4.6.4 options which utilize isotropic thermal and electrical conductivity tensors. Additionally the Energy equation for temperature should have the SRC term active for JOULE HEATING 11.3.13.

Parent Block(s) `ARIA MATERIAL`

4.38.1 SEEBECK COEFFICIENT = CONSTANT

Parameters
```
A = REAL
```

Example
```
SEEBECK COEFFICIENT = CONSTANT A = 1.0
```

Description `A` is the value of the constant Seebeck coefficient. Must be accompanied by the `HEAT_CONDUCTION = THERMOELECTRIC` option.

4.38.2 SEEBECK COEFFICIENT = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Seebeck Coefficient = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.38.3 SEEBECK COEFFICIENT = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Seebeck Coefficient= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$\kappa = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.84}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.38.4 SEEBECK COEFFICIENT = USER_FUNCTION

Parameters

```
NAME = STRING
X = STRING
```

Example

```
begin definition for function SI_K
  type is piecewise linear
  begin values
     20.0     5.50e7
    100.0     4.60e7
     ...
    800.0     1.30e7
   2000.0     1.30e7
  end values
end definition for function SI_K

...

Begin Aria Material Foo
   ...
   Seebeck Coefficient = User_Function Name=SI_K X=Temperature
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the thermal conductivity as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here NAME is the name of the user-defined function for (COEFFICIENT_DATA, SI_K, in the example) and X is the Aria name of the abscissa variable (TEMPERATURE in the example). Note that X is not necessarily the same name as the optional abscissa variable identified in the user-defined function. Must be accompanied by the HEAT_CONDUCTION = THERMOELECTRIC option. Full effect of this contribution can only be realized if Energy equation for temperature has the SRC term active for JOULE HEATING (11.3.13).

4.39 SHELL THICKNESS

SHELL THICKNESS = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the shell thickness.

Summary Specifies the material model and parameters for the shell thickness.

Parent Block(s) ARIA MATERIAL

4.39.1 SHELL THICKNESS = CONSTANT

Parameters T = *REAL*

Example
```
Shell Thickness = Constant T = 0.8
```

Description T is the value of the constant shell thickness.

4.39.2 SHELL THICKNESS = ELEMENT_ATTRIBUTE

Parameters MULTIPLIER = *REAL*

Example
```
Shell Thickness = Element_Attribute multiplier = 1.0
```

Description For shell elements the ExodusII mesh file can contain area attribute values for each element within an element block. This value can then be retrieved and used by the application. This model is used to scale the value of area attribute by Multiplier to obtain the shell thickness for a particular element.

4.40 SKELETON DENSITY

SKELETON DENSITY = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies a model for the porous skeleton density of a material.

Summary Specifies a model for the porous skeleton density of a material.

Parent Block(s) `ARIA MATERIAL`

4.40.1 SKELETON DENSITY = CONSTANT

Parameters `RHO = REAL`

Example `Skeleton Density = Constant Rho=1e3`

Description The value is constant in space and time.

4.40.2 SKELETON DENSITY = ENCORE_FUNCTION

Parameters `NAME = STRING`
`[EVAL_TYPE = STRING]`

Example `Skeleton Density = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.41 SKELETON INTERNAL ENERGY

SKELETON INTERNAL ENERGY = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies a model for the porous skeleton internal energy of a material.

Summary Specifies a model for the porous skeleton internal energy of a material.

Parent Block(s) `ARIA MATERIAL`

4.41.1 SKELETON INTERNAL ENERGY = CONSTANT

Parameters `E = REAL`

Example `Skeleton Internal Energy = Constant E=2.3e-4`

Description The value is constant in space and time.

4.41.2 SKELETON INTERNAL ENERGY = ENCORE_FUNCTION

Parameters `NAME = STRING`
`[EVAL_TYPE = STRING]`

Example `Skeleton Internal Energy = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.41.3 SKELETON INTERNAL ENERGY = LINEAR

Parameters `CP = REAL`
`T_REF = REAL`

Example `Skeleton Internal Energy = Linear Cp=13.7 T_ref=298.15`

Description The internal energy of the porous skeleton, e_s, is given by the simple relation,

$$e_s = C_p \left(T - T_{ref} \right) \tag{4.85}$$

where C_p is the specific heat supplied by the `CP` parameter and T_{ref} is a reference temperature supplied by the `T_REF` parameter. This model also supplies the time derivative of e_s,

$$\frac{\partial e_s}{\partial t} = C_p \frac{\partial T}{\partial t} \tag{4.86}$$

4.42 SKELETON SPECIFIC HEAT

SKELETON SPECIFIC HEAT = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies a model for the porous skeleton specific heat of a material.

Summary Specifies a model for the porous skeleton specific heat of a material.

Parent Block(s) `ARIA MATERIAL`

4.42.1 SKELETON SPECIFIC HEAT = CONSTANT

Parameters `CP = REAL`

Example
```
Skeleton Specific Heat = Constant Cp=400
```

Description The value is constant in space and time.

4.42.2 SKELETON SPECIFIC HEAT = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Skeleton Specific Heat = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.42.3 SKELETON SPECIFIC HEAT = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Skeleton Specific Heat= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$C_p = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.87}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.42.4 SKELETON SPECIFIC HEAT = USER_FUNCTION

Parameters

```
NAME = STRING
X = STRING
```

Example

```
Begin Definition for Function Table_Fcn
   Type is Piecewise Linear
   Begin Values
      273.15  100
      277.15  200
      283.15  300
   End
End
...

Begin Aria Material Foo
   ...
   Skeleton Specific Heat = User_Function Name=Table_Fcn X=Temperature
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the skeleton specific heat as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`Table_Fcn` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.43 SOLID LAMBDA

SOLID LAMBDA = *MODEL* [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the λ Lamé coefficient for the solid stress tensor.

Summary The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk}\boldsymbol{I} + 2\mu\boldsymbol{E} - \beta\left(T - T_{ref}\right)\boldsymbol{I} \tag{4.88}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.89}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu\frac{\nu}{(1-2\nu)} \tag{4.90}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha(3\lambda + 2\mu) \tag{4.91}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Parent Block(s) `ARIA MATERIAL`

4.43.1 SOLID LAMBDA = CONSTANT

Parameters `LAMBDA = REAL`

Example `SOLID LAMBDA = CONSTANT LAMBDA = 1.0`

Description `LAMBDA` is the value of the constant λ.

4.43.2 SOLID LAMBDA = CONVERTED

Parameters `(None)`

Example `SOLID LAMBDA = Converted`

Description Aria will use Young's modulus and Poisson ratio to compute the Lamé λ coefficient. Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

4.43.3 SOLID LAMBDA = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
SOLID LAMBDA = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.44 SOLID TWO MU

SOLID TWO MU = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for twice the μ Lamé coefficient for the solid stress tensor.

Summary The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk} \boldsymbol{I} + 2\mu \boldsymbol{E} - \beta \left(T - T_{ref}\right) \boldsymbol{I} \tag{4.92}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.93}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu \frac{\nu}{(1-2\nu)} \tag{4.94}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha (3\lambda + 2\mu) \tag{4.95}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Parent Block(s) `ARIA MATERIAL`

4.44.1 SOLID TWO MU = CONSTANT

Parameters `TWO_MU = REAL`

Example `SOLID TWO MU = CONSTANT TWO_MU = 1.0`

Description `TWO_MU` is the value of 2μ.

4.44.2 SOLID TWO MU = CONVERTED

Parameters `(None)`

Example `SOLID TWO MU = Converted`

Description Aria will use Young's modulus and Poisson ratio to compute the Lamé μ coefficient. Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

4.44.3 SOLID TWO MU = ENCORE_FUNCTION

Parameters `NAME = STRING`
`[EVAL_TYPE = STRING]`

Example `TWO_MU = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.45 SOLID STRESS

SOLID STRESS = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies a contribution to the solid stress tensor. Multiple stresses are combined additively and may be specified by using this line command multiple times.

Summary Specifies a contribution to the solid stress tensor. The total stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \sum_j \boldsymbol{T}_j \tag{4.96}$$

Parent Block(s) `ARIA MATERIAL`

4.45.1 SOLID STRESS = ISOTHERMAL

Parameters
```
T = REAL
T_REF = REAL
```

Example
```
SOLID STRESS = Isothermal T=500 T_ref=325
```

Description This stress accounts for the mechanical stresses due to thermally induced strains.

$$\boldsymbol{T} = -\beta\left(T - T_{ref}\right)\boldsymbol{I} \tag{4.97}$$

where β is the Lamé coefficient of thermal stress (related to the coefficient of thermal expansion, α), T is the temperature and T_{ref} is the temperature of the undeformed reference state of the mesh (pseudo-solid). This is a specialization of the `THERMAL` model that uses uniform, fixed temperature and thermal strain reference temperature.

4.45.2 SOLID STRESS = LINEAR_ELASTIC

Parameters
```
REFERENCE_FRAME = "MOVING" | "UNDEFORMED"
```

Example
```
SOLID STRESS = Linear_Elastic Reference_Frame=Moving
```

Description Supplies the linear elasticity stress tensor,

$$\boldsymbol{T} = \lambda \operatorname{trace} \boldsymbol{E}\boldsymbol{I} + 2\mu\boldsymbol{E} \tag{4.98}$$

where λ and μ and the Lamé coefficients and $\boldsymbol{E}$ is the strain tensor. When the choice of reference frame is "UNDEFORMED" then the strain is computed in the undeformed reference state; this is commonly referred to as small strain theory. When the reference frame is "MOVING" then the strain is computed with respect to the deformed coordinates.

Specifically, the strain tensor is given by

$$\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^t\right) \tag{4.99}$$

where $\boldsymbol{d}$ is the solid displacement field. The choice of reference frame determines whether the $\boldsymbol{\nabla}$ operator is computed in the undeformed or moving reference frames.

4.45.3 SOLID STRESS = NEOHOOKEAN_ELASTIC

Parameters (none)

Example
```
SOLID STRESS = Neohookean_Elastic
```

Description Supplies a nonlinear hyperelastic stress of the form,

$$\boldsymbol{T} = \frac{\mu}{J}(\boldsymbol{b} - \boldsymbol{I}) + \frac{\lambda}{J}\ln J\boldsymbol{I} \tag{4.100}$$

where λ and μ and the Lamé coefficients, $\boldsymbol{b} \equiv \boldsymbol{F} \cdot \boldsymbol{F}^t$ is the left Cauchy-Green tensor, $\boldsymbol{F}$ is the deformation gradient and $J \equiv \det \boldsymbol{F}$. See, e.g., [4] or [5].

4.45.4 SOLID STRESS = NONLINEAR_ELASTIC

Parameters (none)

Example
```
SOLID STRESS = Nonlinear_Elastic
```

Description Supplies a nonlinear elastic stress,

$$\boldsymbol{T} = \lambda \operatorname{trace} \boldsymbol{E}\boldsymbol{I} + 2\mu\boldsymbol{E} \tag{4.101}$$

where λ and μ and the Lamé coefficients and $\boldsymbol{E}$ is the strain tensor. The particular choice of strain tensor chosen depends on where the configuration (reference frame) which is set via the MESH MOTION command line. See section 5.16 for more information. When the MESH MOTION is set to REFERENCE_FRAME then the Green strain is used. Otherwise, the Almansi strain is used.

4.45.5 SOLID STRESS = RESIDUAL

Parameters
```
[SXX | SX = REAL]
[SYY | SY = REAL]
[SZZ | SZ = REAL]
[SXY = REAL]
[SXZ = REAL]
[SYZ = REAL]
```

Example
```
SOLID STRESS = Residual Sxx=0.02 Syy=0.02
```

Description This stress accounts for the initial residual stress in a solid that is constant and uniform everywhere. The components of the residual stress tensor are supplied by the (up-to) six components `SXX`, `SYY`, `SZZ`, `SXY`, `SXZ`, and `SYZ`.

This is directly analogous to the `ISTRESS` condition in ANSYS. To that end, the diagonal components can be specified as either `SXX` or SX etc.

4.45.6 SOLID STRESS = THERMAL

Parameters `(none)`

Example `SOLID STRESS = Thermal`

Description This stress accounts for the mechanical stresses due to thermally induced strains.

$$\boldsymbol{T} = -\beta\,(T - T_{ref})\,\boldsymbol{I} \tag{4.102}$$

where β is the Lamé coefficient of thermal stress (related to the coefficient of thermal expansion, α), T is the temperature and T_{ref} is the temperature of the undeformed reference state of the solid.

4.45.7 SOLID STRESS = LAME

Parameters `MODEL = STRING`

Example `SOLID STRESS = Lame Model = Neo_Hookean`

Description This stress interfaces with the Lame constitutive model library used by Sierra/SolidMechanics. As such, it allows the use of many of Lame's constitutive models. Sierra/Multimechanics prepares the kinematic quantities, and Lame returns the Cauchy stress, which is integrated in the `SOLID` equation.

Material parameters for the constitutive models are handled by an input block consistent with the Sierra/SolidMechanics formatting. For documentation of the available constitutive models and the required parameter input, please see the Sierra/SolidMechanics documentation.

Currently this interface is instrumented to support hyperelastic constitutive models, and is restricted to the following models:

- `NEO_HOOKEAN`
- `THERMALBATTERYSEPARATOR`
- `SOILFOAMPHASETRANSITION`

4.46 SPECIES DIFFUSION

SPECIES DIFFUSION = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material (constitutive) model for the species diffusion (diffusive flux) in the bulk.

Summary Specifies the material (constitutive) model for the species diffusion (diffusive flux) in the bulk.

Parent Block(s) `ARIA MATERIAL`

4.46.1 SPECIES DIFFUSION = BASIC

This is an alias for `FICKS_LAW`.

Example `Species Diffusion = Basic`

4.46.2 SPECIES DIFFUSION = FICKS_LAW

Parameters `(none)`

Example `Species Diffusion = Ficks_Law`

Description The diffusive species flux $\boldsymbol{q}$ is given by Fick's Law,

$$\boldsymbol{q} = -D\boldsymbol{\nabla}C \tag{4.103}$$

where D is the species diffusivity and C is the species concentration.

4.47 SPECIES DIFFUSIVITY

SPECIES DIFFUSIVITY = *MODEL* [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the species diffusivity.

Summary Specifies the material model for the species diffusivity.

Parent Block(s) `ARIA MATERIAL`

4.47.1 SPECIES DIFFUSIVITY = ARRHENIUS

Parameters
```
[C = REAL]
D = REAL
Q = REAL
R = REAL
```

Example `SPECIES DIFFUSIVITY = Arrhenius D=2e-6 Q=6900 R=8.314`

Description The species diffusivity is given by the Arrhenius function

$$D = C + D_\circ e^{-Q/RT} \tag{4.104}$$

where C is given by the `C` $D_\circ$ is given by the `D` parameter, Q is given by the `Q` parameter and R is given by the `R` parameter.

4.47.2 SPECIES DIFFUSIVITY = CONSTANT

Parameters `D = REAL`

Example `SPECIES DIFFUSIVITY = CONSTANT D = 1.0`

Description `D` is the value of the constant species diffusivity.

4.47.3 SPECIES DIFFUSIVITY = ENCORE_FUNCTION

Parameters `NAME = STRING`
`[EVAL_TYPE = STRING]`

Example `SPECIES DIFFUSIVITY = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.48 SPECIFIC HEAT

SPECIFIC HEAT = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the specific heat.

Summary Specifies the material model for the specific heat.

Parent Block(s) `ARIA MATERIAL`

4.48.1 SPECIFIC HEAT = CONSTANT_EVALUATOR

Parameters `CP = REAL h0 = REAL`

Example SPECIFIC HEAT = CONSTANT_EVALUATOR CP = 1.0 h0 = 500

Description CP is the value of the constant specific heat, and h0 (optional) is the value of the reference enthalpy. This also creates an expression for enthalpy, as $h(T) = h_0 + C_p T$.

4.48.2 SPECIFIC HEAT = POLYNOMIAL_EVALUATOR

Parameters a0 = *REAL* a1 = *REAL* a2 = *REAL* a3 = *REAL* a4 = *REAL* a5 = *REAL* a6 = *REAL* a7 = *REAL* a8 = *REAL* R = *REAL*

Example SPECIFIC HEAT = POLYNOMIAL_EVALUATOR a0 = 50 a1 = 1000 a2 = 1 R = 8.314

Description The coefficients a0 to a8 are the polynomial coefficients and R is the gas constant they are based on (defaults to 8.314). Only a0 and a1 are required.

Specific heat is evaluated using $C_p(T)/R = \sum_{i=1}^{n} a_i T^{i-1}$ and enthalpy by $h(T)/R = a_0 + \sum_{i=1}^{n} \frac{1}{i} a_i T^i$. The polynomial order is determined by how many non-zero entries are provided.

4.48.3 SPECIFIC HEAT = TABULAR_EVALUATOR

Parameters Cp_function = *STRING* h0 = *REAL*

Example SPECIFIC HEAT = TABULAR_EVALUATOR Cp_function = f_Cp

Description The function specified here is the same syntax as for USER_FUNCTION, but the function must be with respect to temperature. The reference enthalpy, h0 is optional. This also provides an expression for enthalpy that is based on an integration of the tabular values of specific heat.

4.48.4 SPECIFIC HEAT = T_EXPONENT

Parameters Cp_ref = *REAL* T_ref = *REAL* n = *REAL* h0 = *REAL*

Example SPECIFIC HEAT = T_EVALUATOR Cp_ref = 1000 T_ref = 300 n = 2

Description This provides specific heat as $C_p(T) = C_{p,ref} \left(\frac{T}{T_{ref}} \right)^n$ and enthalpy as $h(T) = h_0 + \frac{C_{p,ref}}{T_{ref}^n} \frac{T^{n+1}}{n+1}$.

4.48.5 SPECIFIC HEAT = CONSTANT

Parameters CP = *REAL*

Example SPECIFIC HEAT = CONSTANT CP = 1.0

Description CP is the value of the constant specific heat.

4.48.6 SPECIFIC HEAT = CURING_FOAM

Parameters
```
VFRAC_SUBINDEX = INT
[CP_FL = REAL]
[CP_FG = REAL]
[CP_E = REAL]
[PHI_ZERO = REAL]
```

Example
```
Specific Heat = Curing_Foam Vfrac_Subindex=1 Cp_fL=1 Cp_fG=1 Cp_e=1
phi_zero=0.2
```

Description For a curing epoxy with volume fraction ϕ the specific heat is given by

$$C_p = C_{p,f_L}\phi + C_{p,f_G}(\phi_\circ - \phi) + C_{p,e}(1-\phi_\circ) \quad (4.105)$$
$$= a + b\phi \quad (4.106)$$

where C_{p,f_L} is the specific heat of the liquid phase flourinert, C_{p,f_G} is the specific heat of the gas phase flourinert, $C_{p,e}$ is the specific heat of the epoxy and $\phi_\circ$ is the reference volume fraction in the flourinert. In the latter form of this relationship

$$a = C_{p,f_L} - C_{p,f_G} \quad (4.107)$$
$$b = C_{p,f_G}\phi_\circ + C_{p,e}(1-\phi_\circ). \quad (4.108)$$

NOTE: The volume fraction is assumed to be a SPECIES field with the subindex provided by the VFRAC_SUBINDEX parameter.

4.48.7 SPECIFIC HEAT = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
SPECIFIC HEAT = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument EVAL_TYPE can be used to select the particular function on the Encore function object. Valid options include VALUE, DOT, GRADIENT, FLUX and STRESS. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – VALUE for NO_OP expressions, DOT for DT_OP expressions and GRADIENT for GRAD_OP expressions.

4.48.8 SPECIFIC HEAT = EXPONENTIAL

Parameters
```
VARIABLE = STRING
[CONSTANT = REAL]
[MULTIPLIER = REAL]
EXPONENT = REAL
```

Example
```
Specific Heat = Exponential Variable=Temperature Multiplier=1.0
Exponent=-0.3
```

Description Exponential function of in specified scalar variable. The specific heat is computed as

$$C_p = C + Me^{EX} \tag{4.109}$$

Here, C is the constant term supplied by the `CONSTANT` parameter which defaults to zero, M is the value supplied by the `MULTIPLIER` parameter which defaults to unity, X is the variable supplied by the `VARIABLE` parameter and E is the exponential multiplier provided by the `EXPONENT` parameter.

4.48.9 SPECIFIC HEAT = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Specific Heat= Polynomial Variable=Temperature Order=1 C0=401.0 C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$C_p = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.110}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.48.10 SPECIFIC HEAT = USER_FUNCTION

Parameters
```
NAME = STRING
X = STRING
```

Example

```
begin definition for function Water_Heat_Capacity
   # Source Appendix 2 from "Transport Processes and
   # Unit Operations" by C. J. Geankoplis
   type is piecewise linear
   begin values
      # K     J / kg K
      273.15  4220
      283.15  4195
      293.15  4185
      298.15  4182
      303.15  4181
      313.15  4181
      323.15  4183
      333.15  4187
      343.15  4192
      353.15  4199
      363.15  4208
      373.15  4219
   end
end
...

Begin Aria Material Foo
   ...
   Specific Heat = User_Function X=Temperature Name=Water_Heat_Capacity
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the specific heat as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`Water_Heat_Capacity` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.48.11 SPECIFIC HEAT = USE_PHASE_CHANGE

SPECIFIC HEAT = USE_PHASE_CHANGE FLH=LH_VAL TS=T_SOLIDUS TL=T_LIQUIDUS

Description

Defines a specialized material model to account for change of phase during melting or solidification. Must be accompanied by the PHASE CHANGE SPECIFIC HEAT command line.

Summary

Phase change is being modeled by treating the phase transition temperature region as a mushy-zone about the melt temperature. Here the energy associated with phase change in the mushy-zone is accounted for by modifying the specific heat to include the latent heat of fusion as an additional contribution to the energy equation.

$$\text{Specific Heat} = \begin{cases} C_p(T) & T_s < T \\ C_p(T) + \frac{FLH}{T_l - T_S} & T_s < T < T_l \\ C_p(T) & T > T_l \end{cases}$$

where T is the current temperature, $C_p(T)$ is the specific heat of the material, FLH is the latent heat of fusion, T_s is a solidus temperature, T_l is a liquidus temperature and $T_l - T_s$ is the temperature range of the mushy zone.

Material outside of the mushy-zone, $T < T_s$ or $T > T_l$, will employ a standard specific heat model, $C_p(T)$.

While a material is undergoing phase change, the time stepping will be restricted so that the modeled phase change behavior will be captured. Use of this model will often require additional restrictions be placed upon the adaptive time stepping 16.4, 16.4.1.

Parent Block(s) `ARIA MATERIAL`

Parameters
FLH = *REAL*
T_S = *REAL*
T_L = *REAL*

Example
```
Specific Heat = USE_PHASE_CHANGE FLH = 2.3E+04 TS = 599 TL = 600.5
```

4.49 PHASE CHANGE SPECIFIC HEAT

PHASE CHANGE SPECIFIC HEAT = SPECIFIC_HEAT_MODEL [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Defines the specific heat material model used in conjunction with phase change. This command can request any valid SPECIFIC HEAT model aside from USE_PHASE_CHANGE and must be accompanied by a SPECIFIC HEAT = USE_PHASE_CHANGE command line.

Summary Includes the parameters for the specific heat material model.

Parent Block(s) `ARIA MATERIAL`

Parameters `The optional parameters are those appropriate to the SPECIFIC_HEAT_MODEL.`

Example
```
Phase Change Specific Heat = Constant cp = 1.0
```

4.50 SURFACE TENSION

SURFACE TENSION = *MODEL* [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the model to use for the surface (interfacial) tension.

Summary Specifies the model to use for the surface (interfacial) tension.

Parent Block(s) `ARIA MATERIAL`

4.50.1 SURFACE TENSION = CONSTANT

Parameters `SIGMA = REAL`

Example `Surface Tension = Constant Sigma = 72.0`

Description `SIGMA` is the value of the surface tension.

4.50.2 SURFACE TENSION = ENCORE_FUNCTION

Parameters `NAME = STRING`
`[EVAL_TYPE = STRING]`

Example `Surface Tension = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.50.3 SURFACE TENSION = LINEAR_T

Parameters `SIGMA0 = REAL`
`DSIGMADT = REAL`
`T_REF = REAL`

Example `Surface Tension = Linear_T sigma0=72.  dsigmadT = -.15 T_ref = 298.`

Description `SIGMA0` is the value of the surface tension at the reference temperature `T_REF` and `DSIGMADT` is the derivative of the surface temperature with respect to temperature, i.e.,

$$\sigma = \sigma_0 + m(T - T_{ref}) \tag{4.111}$$

where m is `DSIGMADT`.

4.51 SUSPENSION FLUX

SUSPENSION FLUX = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the parameters for the suspension flux model.

Summary Specifies the suspension flux model and its parameters for this material.

Parent Block(s) `ARIA MATERIAL`

4.51.1 SUSPENSION FLUX = PHILLIPS

Parameters
```
K_mu = REAL
K_c = REAL
phi_max = REAL
beta = REAL
particle_radius = REAL
```

Example
```
SUSPENSION FLUX = Phillips K_mu=0.62 K_c=0.41 phi_max=0.68 beta=-1.82
particle_radius=0.01
```

Description The Phillips diffusive flux model is intended to be used in conjunction with the Krieger viscosity model (4.56.10). Here, the flux is given by

$$\boldsymbol{q} = \left(K_c a^2 - K_\mu a^2 \beta \frac{\phi}{\phi_m - \phi} \right) \dot{\gamma} \phi \boldsymbol{\nabla} \phi + K_c a^2 \phi^2 \boldsymbol{\nabla} \dot{\gamma} \tag{4.112}$$

where $\dot{\gamma}$ is the shear rate, ϕ is the suspension concentration, ϕ_m is the maximum suspension concentration and a is the particle radius.

4.52 THERMAL CONDUCTIVITY

THERMAL CONDUCTIVITY = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the thermal conductivity.

Summary Specifies the material model for the thermal conductivity that appears in the diffusion term of the energy equation for temperature. The command line must be accompanied by either the `HEAT_CONDUCTION = BASIC` option for an isotropic conductivity tensor or the `HEAT_CONDUCTION = GENERALIZED` option for the anisotropic case.

Parent Block(s) `ARIA MATERIAL`

4.52.1 THERMAL CONDUCTIVITY = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example
```
Thermal Conductivity = Calore_User_Sub name=w80afftuser type=element NR=2
R0=1000 R1=0.7
```

Description The Calore_User_Sub model allows the user to explicitly evaluate thermal conductivity as a function of temperature and position within an element. Additionally, a user_query feature allows evaluation of thermal conductivity based upon other problem criteria. `NAME` is the name of the user subroutine was registered with. `NR` and `NI` are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the `R`n (`I`n) parameter. **Note:** the parameters use zero based counting. The `TYPE` parameter denotes the API the user subroutine uses; currently Aria supports type `ELEMENT` for scalar valued thermal conductivity.

The optional `MULTIPLIER` parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

Must be accompanied by the `HEAT_CONDUCTION` = `BASIC` option.

4.52.2 THERMAL CONDUCTIVITY = CONSTANT

Parameters
```
K = REAL
```

Example
```
THERMAL CONDUCTIVITY = CONSTANT K = 1.0
```

Description `K` is the value of the constant thermal conductivity. Must be accompanied by the `HEAT_CONDUCTION` = `BASIC` option.

4.52.3 THERMAL CONDUCTIVITY = CURING_FOAM

Parameters
```
RHO_E = REAL
K_F = REAL
K_E = REAL
```

Example
```
Thermal Conductivity = Curing_Foam rho_e=1.3 k_e=14 k_f=2.7
```

Description For a curing epoxy with mixture density ρ

$$\kappa = \frac{2}{3}\left(\frac{\rho}{\rho_e}\right)\kappa_e + \left(1 - \frac{\rho}{\rho_e}\right)\kappa_f \tag{4.113}$$

$$= a + b\rho \tag{4.114}$$

where ρ_e is the density of the epoxy, κ_e is the thermal conductivity of the epoxy and κ_f is the thermal conductivity of the flourinert. In the latter form of this relationship

$$a = \kappa_f \tag{4.115}$$

$$b = \frac{1}{\rho_e}\left(\frac{2}{3}\kappa_e - \kappa_f\right) \tag{4.116}$$

Must be accompanied by the `HEAT_CONDUCTION` = `BASIC` option.

4.52.4 THERMAL CONDUCTIVITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Thermal Conductivity = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.52.5 THERMAL CONDUCTIVITY = MESH_INPUT

Parameters
```
NAME = STRING
AXIS1 = STRING
AXIS2 = STRING
[TYPE = STRING]
[MULTIPLIER = REAL]
[TOLERANCE = REAL]
```

Example
```
THERMAL CONDUCTIVITY = MESH_INPUT NAME = COND AXIS1 = A1 AXIS2 = A2
TOLERANCE = 1.0E-4 TYPE = SYMMETRIC MULTIPLIER = 2.0
```

Description

Thermal conductivity is being provided from the input mesh file.

The TYPE parameter defines what tensor components are supplied in the input file. The default TYPE = PRINCIPAL will cause the code to expect principal components only. Supplying TYPE = SYMMETRIC will require that the file contain corresponding the symmetric tensor components.

The NAME parameter specifies the base name of the tensor component values present in the mesh file. Component names are internally generated using the base name and the component subscripts. For example for TYPE = SYMMETRIC in the two-dimensional a base name of COND would generate COND11, COND22 and COND12, and for TYPE = PRINCIPAL would generate COND11 and COND22.

The tensor components provided are assumed to be defined in an orientation compactly described with unit vectors. For two-dimensional problems one unit vector must be provided and for three-dimensional problems one must provide two unit vectors. In 3D the AXIS1 and AXIS2 parameters define the variable names which specify orientation vectors associated with the components. Default value of AXIS1 = FIBER and default value of AXIS2 = NORMAL. Orientation vectors are given relative to the fixed Cartesian (xyz) frame. Internal to the code AXIS1 and AXIS2 will be used to generate the remaining orthogonal axis needed to define a complete ortho-normal transformation into the computational reference frame. Internal checks are done to guarantee that orientation vectors are of nonzero length. For 2D only the AXIS1 variable need be provided and AXIS2, if provided, will be ignored.

The TOLERANCE parameter defines a desired tolerance for the dot product of vectors defined by AXIS1 and AXIS2. An error message will be generated when the dot product is greater than the specified tolerance. Default value for TOLERANCE = 1.0E9

Use of the MULTIPLIER parameter results in scaling all components of the conductivity tensor.

4.52.6 THERMAL CONDUCTIVITY = OPTICALLY_THICK

Parameters

```
K = REAL
N = REAL
BETA_R = REAL
```

Example

```
THERMAL CONDUCTIVITY = OPTICALLY_THICK K = 3.0 N = 2.1 BETA_R = 1.0e-5
```

Description

Thermal conductivity used to model combined pure diffusion and radiative diffusion in the optically-thick limit. Constant values for the pure diffusion thermal conductivity K, index of refraction n, and Rosseland-mean extinction coefficient β_R must be provided on the input line.

$$\kappa = K + \frac{16\sigma n^2}{3\beta_R} T^3 \tag{4.117}$$

Must be accompanied by the `HEAT_CONDUCTION = BASIC` option. Additionally the `GLOBAL CONSTANTS` block must include include the `STEFAN-BOLTZMANN CONSTANT` entry to provide a value of σ.

4.52.7 THERMAL CONDUCTIVITY = POWER_LAW

Parameters

```
A = REAL
GAMMA = REAL
```

Example `THERMAL CONDUCTIVITY = POWER_LAW A = 3.8 GAMMA = -1.2`

Description Thermal conductivity is a power law function of temperature

$$\kappa = AT^{\gamma} \tag{4.118}$$

where A is a scale factor and γ is the power law exponent. For values of $\gamma = 1$ the THERMAL model 4.52.9 should be used instead. Must be accompanied by the `HEAT_CONDUCTION` = `BASIC` option.

4.52.8 THERMAL CONDUCTIVITY = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Thermal Conductivity= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$\kappa = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.119}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.52.9 THERMAL CONDUCTIVITY = THERMAL

Parameters
```
[A = REAL]
[B = REAL]
[C = REAL]
[D = REAL]
```

Example `THERMAL CONDUCTIVITY = THERMAL A = 401.0 B = 88.5`

Description Cubic polynomial function of temperature for the conductivity.

$$\kappa = A + BT + CT^2 + DT^3 \tag{4.120}$$

Must be accompanied by the `HEAT_CONDUCTION` = `BASIC` option.

4.52.10 THERMAL CONDUCTIVITY = USER_FUNCTION

Parameters

```
NAME = STRING
X = STRING
```

Example

```
begin definition for function SI_K
  type is piecewise linear
  begin values
     20.0     5.50e7
    100.0     4.60e7
    ...
    800.0     1.30e7
   2000.0     1.30e7
  end values
end definition for function SI_K

...

Begin Aria Material Foo
   ...
   Thermal Conductivity = User_Function Name=SI_K X=Temperature
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the thermal conductivity as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function for (`CONDUCTIVITY_DATA`, SI_K, in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the optional abscissa variable identified in the user-defined function. Must be accompanied by the `HEAT_CONDUCTION` = `BASIC` option.

4.52.11 THERMAL CONDUCTIVITY = CONSTANT (TENSOR)

Parameters

```
XX = REAL= REAL
[XY = REAL]
[XZ = REAL]
[YX = REAL]
YY = REAL= REAL
[YZ = REAL]
[ZX = REAL]
[ZY = REAL]
[ZZ = REAL]
```

Example

```
TENSOR THERMAL CONDUCTIVITY = Constant XX=1.0 YY=2.0 ZZ=1.0
TENSOR THERMAL CONDUCTIVITY = Constant 11=1.0 22=2.0 33=1.0
```

Description

All components default to zero and all values are constant in space and time. Must be accompanied by the `HEAT_CONDUCTION` = `GENERALIZED` option.

4.52.12 THERMAL CONDUCTIVITY = USER_FUNCTION (TENSOR)

Parameters

```
NAME_XX = STRING
NAME_YY = STRING
[NAME_ZZ = STRING]
X = STRING
```

Example

```
begin definition for function SI_K11
  type is piecewise linear
  begin values
     20.0     5.50e7
    100.0     4.60e7
    ...
    800.0     1.30e7
   2000.0     1.30e7
  end values
end definition for function SI_K11

begin definition for function SI_K22
  type is piecewise linear
  begin values
     20.0     8.50e7
    100.0     7.60e7
    ...
    800.0     0.30e7
   2000.0     0.30e7
  end values
end definition for function SI_K22

...

Begin Aria Material Foo
   ...
   Tensor Thermal Conductivity = User_Function Name_xx=SI_K11 Name_xx=SI_K22 X=Temperature
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the thermal conductivity as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME_XX`, `NAME_YY`, `NAME_ZZ`, are the names of user-defined functions for (`CONDUCTIVITY_DATA`, SI_K11, SI_K22 in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). `NAME_ZZ` is required only for three-dimensional analysis. Note that `X` is not necessarily the same name as the optional abscissa variable identified in the user-defined function. Must be accompanied by the `HEAT_CONDUCTION = GENERALIZED` option.

4.52.13 THERMAL CONDUCTIVITY = CALORE_USER_SUB (TENSOR)

Parameters

```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example

```
Tensor Thermal Conductivity = Calore_User_Sub name=w80afftuser
type=element_tensor NR=2 R0=1000 R1=0.7
```

Description

The Calore_User_Sub Tensor model allows the user to explicitly evaluate thermal conductivity as a function of temperature and position within an element. Additionally, a user_query feature allows evaluation of thermal conductivity based upon other problem criteria. `NAME` is the name of the user subroutine was registered with. `NR` and `NI` are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the `R`n (`I`n) parameter. **Note:** the parameters use zero based counting. The `TYPE` parameter denotes the API the user subroutine uses; currently Aria supports type `ELEMENT_TENSOR` for tensor valued thermal conductivity.

The optional `MULTIPLIER` parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

Must be accompanied by the `HEAT_CONDUCTION` = `GENERALIZED` option.

4.53 THERMAL DIFFUSIVITY

THERMAL DIFFUSIVITY = *MODEL* [param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description

Specifies the material model for the thermal diffusivity.

Summary

Specifies the material model for the thermal diffusivity.

Parent Block(s) `ARIA MATERIAL`

4.53.1 THERMAL DIFFUSIVITY = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example
```
Thermal Diffusivity = Calore_User_Sub name=w80afftuser type=element NR=2
R0=1000 R1=0.7
```

Description `NAME` is the name of the user subroutine was registered with. `NR` and `NI` are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the `R`n (`I`n) parameter. **Note:** the parameters use zero based counting. The `TYPE` parameter denotes the API the user subroutine uses; currently Aria only supports types `ELEMENT` and `NODE`.

The optional `MULTIPLIER` parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

4.53.2 THERMAL DIFFUSIVITY = CONSTANT

Parameters
```
D = REAL
```

Example
```
THERMAL DIFFUSIVITY = CONSTANT D = 1.0
```

Description `D` is the value of the constant thermal diffusivity.

4.53.3 THERMAL DIFFUSIVITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
THERMAL DIFFUSIVITY = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.54 TOTAL INTERNAL ENERGY

TOTAL INTERNAL ENERGY = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies a model for the total internal energy of a material.

Summary Specifies a model for the total internal of a material.

Parent Block(s) `ARIA MATERIAL`

4.54.1 TOTAL INTERNAL ENERGY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Total Internal Energy = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.54.2 TOTAL INTERNAL ENERGY = POROUS

Parameters `(none)`

Example
```
Total Internal Energy = Porous
```

Description The total internal energy e is computed as

$$e = (1 - \phi)\rho_s e_s + \phi * \rho_f e_f \tag{4.121}$$

where ϕ is the porosity, ρ_s is the density of the solid porous skeleton, e_s is the internal energy of the solid porous skeleton, ρ_f is the density of the fluid phase and e_f is the internal energy of the fluid phase.

4.55 VALENCE

VALENCE = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the valence (net charge) for a species.

Summary Specifies the valence (net charge) for a species.

Parent Block(s) `ARIA MATERIAL`

4.55.1 VALENCE = CONSTANT

Parameters
```
Z = REAL
[SUBINDEX = INT]
```

Example
```
Valence = Constant Subindex=1 Z = 1
Valence = Constant Subindex=2 Z = -1
Valence = Constant Subindex=5 Z = -2
```

Description `Z` is the value of the species valence.

`SUBINDEX` is the species subindex.

4.55.2 VALENCE = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
Valence = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.56 VISCOSITY

VISCOSITY = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the fluid viscosity.

Summary Specifies the material model for the fluid viscosity.

Parent Block(s) `ARIA MATERIAL`

4.56.1 VISCOSITY = ARRHENIUS

Parameters
```
mu0 = REAL
E = REAL
```

Example
```
VISCOSITY = Arrhenius mu0=16.4 E=5000.
```

Description This model provides a viscosity with an Arrhenius temperature dependence:

$$\mu = \mu_0 e^{-E/T} \tag{4.122}$$

where T is the temperature.

4.56.2 VISCOSITY = ARRHENIUS_CARREAU

Parameters
```
MU_ZERO = REAL
[MU_INF = REAL]
[A = REAL]
N = REAL
[LAMBDA = REAL]
K = REAL
T_REF = REAL
[SKIP_SENSITIVITIES = INT]
```

Example
```
Viscosity = Arrhenius_Carreau ...
```

Description This viscosity model is a function of the shear rate (Carreau model) and temperature (Arrhenius model). The two contributions are combined multiplicatively.

$$\mu = \left[\mu_\infty + (\mu_\circ - \mu_\infty)\left(1 + (\lambda\dot{\gamma})^a\right)^{\frac{n-1}{a}}\right] e^{\frac{K}{T} - \frac{K}{T_{ref}}} \tag{4.123}$$

where μ_∞ is the infinite shear viscosity (MU_INF, defaults to zero), $\mu_\circ$ is the zero shear viscosity (MU_ZERO), n (N) and a (A, defaults to 2) are model parameters, $\dot{\gamma}$ is the shear rate, λ (LAMBDA) is a time constant (defaults to 1), T_{ref} is a reference temperature and K is an Arrhenius constant.

If the optional flag SKIP_SENSITIVITIES is nonzero then no Newton sensitivities will be included.

4.56.3 VISCOSITY = BINGHAM_WLF

Parameters

```
MU_ZERO = REAL
MU_INF = REAL
F = REAL
N = REAL
A = REAL
LAMBDA = REAL
TAU_Y = REAL
[SKIP_SENSITIVITIES = INT]
```

Example

```
VISCOSITY = Bingham_WLF ...
```

Description

$$\mu = \mu_\infty + \left(\mu_\circ - \mu_\infty + \tau_y \frac{1 - e^{-\dot{\gamma}F}}{\dot{\gamma}}\right)\left(1 + (\lambda\dot{\gamma})^a\right)^{\frac{n-1}{a}} \tag{4.124}$$

where $\dot{\gamma}$ is the shear rate.

If the optional flag SKIP_SENSITIVITIES is nonzero then no Newton sensitivities will be included.

4.56.4 VISCOSITY = BINGHAM_WLFT

Parameters

```
MU_ZERO = REAL
MU_INF = REAL
F = REAL
N = REAL
A = REAL
LAMBDA = REAL
TAU_Y = REAL
C_1 = REAL
C_2 = REAL
T_REF = REAL
[SKIP_SENSITIVITIES = INT]
```

Example

```
VISCOSITY = Bingham_WLFT ...
```

Description

$$\mu = a_T\left(\mu_\infty + \left(\mu_\circ - \mu_\infty + \tau_y \frac{1 - e^{-a_T\dot{\gamma}F}}{a_T\dot{\gamma}}\right)\left(1 + (a_T\lambda\dot{\gamma})^a\right)^{\frac{n-1}{a}}\right) \tag{4.125}$$

where

$$a_T = e^{\frac{c_1(T_\circ - T)}{c_2 + T - T_\circ}} \tag{4.126}$$

and T is the temperature and $\dot{\gamma}$ is the shear rate.

If the optional flag SKIP_SENSITIVITIES is nonzero then no Newton sensitivities will be included.

4.56.5 VISCOSITY = CARREAU

Parameters
```
MU_ZERO = REAL
[MU_INF = REAL]
[A = REAL]
N = REAL
LAMBDA = REAL
[SKIP_SENSITIVITIES = INT]
```

Example
```
Viscosity = Carreau ...
```

Description

$$\frac{\mu - \mu_\infty}{\mu_\circ - \mu_\infty} = \left(1 + (\lambda\dot{\gamma})^a\right)^{\frac{n-1}{a}} \tag{4.127}$$

or

$$\mu = \mu_\infty + (\mu_\circ - \mu_\infty)\left(1 + (\lambda\dot{\gamma})^a\right)^{\frac{n-1}{a}} \tag{4.128}$$

where μ_∞ is the infinite shear viscosity (MU_INF, defaults to zero), $\mu_\circ$ is the zero shear viscosity (MU_ZERO), n (N) and a (A, defaults to 2) are model parameters, $\dot{\gamma}$ is the shear rate and λ (LAMBDA) is a time constant.

If the optional flag SKIP_SENSITIVITIES is nonzero then no Newton sensitivities will be included.

4.56.6 VISCOSITY = CARREAU_T

Parameters
```
MU_ZERO = REAL
[MU_INF = REAL]
[A = REAL]
N = REAL
K = REAL
[SKIP_SENSITIVITIES = INT]
```

Example
```
Viscosity = Carreau_T ...
```

Description

$$\frac{\mu - \mu_\infty}{\mu_\circ - \mu_\infty} = \left(1 + \left(e^{k/T}\dot{\gamma}\right)^a\right)^{\frac{n-1}{a}} \tag{4.129}$$

or

$$\mu = \mu_\infty + (\mu_\circ - \mu_\infty)\left(1 + \left(e^{k/T}\dot{\gamma}\right)^a\right)^{\frac{n-1}{a}} \tag{4.130}$$

where μ_∞ is the infinite shear viscosity (MU_INF, defaults to zero), $\mu_\circ$ is the zero shear viscosity (MU_ZERO), n (N) and a (A, defaults to 2) are model parameters and $\dot{\gamma}$ is the shear rate. The quantity $e^{k/T}$, where T is temperature and k (K) is a reference temperature, is a temperature dependent time scale; it takes the place of the constant λ time scale in the CARREAU model.

If the optional flag SKIP_SENSITIVITIES is nonzero then no Newton sensitivities will be included.

4.56.7 VISCOSITY = CONSTANT

Parameters
```
MU = REAL
```

Example
```
VISCOSITY = CONSTANT MU = 1.0
```

Description `MU` is the value of the constant fluid viscosity.

4.56.8 VISCOSITY = CURING_FOAM

Parameters
```
VFRAC_SUBINDEX = INT
EXTENT_SUBINDEX = INT
PHI_ZERO = REAL
[A = REAL]
[B = REAL]
[C = REAL]
[KSI_C = REAL]
```

Example
```
Viscosity = Curing_Foam Vfrac_Subindex=1 Extent_Subindex=2 Phi_Zero=0.45
```

Description For a curing epoxy with volume fraction ϕ and extent of reaction ξ the viscosity is given by

$$\mu = \mu_\circ \exp \frac{\phi_\circ - \phi}{1 - \phi_\circ + \phi} \tag{4.131}$$

where $\mu_\circ$ is given by

$$\mu_\circ = (a - bT) \left(\frac{\xi_c^2 - \xi^2}{\xi_c^2} \right)^c \tag{4.132}$$

where T is the temperature. The remaining parameters a, b, c and ξ_c have default values of $a = 20$, $b = 0.22$, $c = -4/3$ and $\xi_c = 0.45$ though they can be overridden with the optional model parameters.

NOTE: The volume fraction is assumed to be a `SPECIES` field with the subindex provided by the `VFRAC_SUBINDEX` parameter. Likewise, the extent of reaction field is assumed to be a `SPECIES` field with the subindex provided by the `EXTENT_SUBINDEX` parameter.

4.56.9 VISCOSITY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
VISCOSITY = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.56.10 VISCOSITY = KRIEGER

Parameters

```
BETA = REAL
PHI_MAX = REAL
MU_S = REAL
```

Example

```
VISCOSITY = KRIEGER BETA = -1.65, PHI_MAX = 1.0, MU_S = 1.0
```

Description

In the viscosity model of [6]

$$\mu = \mu_s \left(1 - \frac{\phi}{\phi_m}\right)^{\beta} \tag{4.133}$$

`BETA` is the Krieger exponent, `PHI_MAX` is the maximum suspension concentration and `MU_S` is the solvent viscosity.

4.56.11 VISCOSITY = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Viscosity= Polynomial Variable=Temperature Order=1 C0=401.0 C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$\mu = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.134}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.56.12 VISCOSITY = POWER_LAW

Parameters
```
K = REAL
N = REAL
```

Example
```
Viscosity = Power_Law K=0.8 N=0.5
```

Description The viscosity is proportional to the shear rate, $\dot{\gamma}$ raised to some power, e.g.,

$$\mu = k\dot{\gamma}^n \tag{4.135}$$

where k (K) and n (N) are model parameters.

4.56.13 VISCOSITY = THERMAL

Parameters
```
[A = REAL]
[B = REAL]
[C = REAL]
[D = REAL]
```

Example
```
VISCOSITY = THERMAL A=1750 C=0.12 D=0
```

Description This model is simply a cubic polynomial in temperature where the viscosity is given by

$$\mu = A + BT + CT^2 + DT^3 \tag{4.136}$$

where T is the temperature.

4.56.14 VISCOSITY = USER_FUNCTION

Parameters
```
NAME = STRING
X = STRING
```

Example

```
begin definition for function Water_Viscosity
   # Source Appendix 2 from "Transport Processes and
   # Unit Operations" by C. J. Geankoplis
   type is piecewise linear
   begin values
      # K      Pa*s (or cP)
      273.15  1.7921
      275.15  1.6728
      277.15  1.5674
      279.15  1.4728
      281.15  1.3860
      283.15  1.3077
      285.15  1.2363
      287.15  1.1709
      289.15  1.1111
      291.15  1.0559
      293.15  1.0050
      293.35  1.0000
      295.15  0.9579
      297.15  0.9142
      299.15  0.8737
      301.15  0.8360
      303.15  0.8007
      305.15  0.7679
      307.15  0.7371
      309.15  0.7085
      311.15  0.6814
      313.15  0.6560
      315.15  0.6321
      317.15  0.6097
      319.15  0.5883
      321.15  0.5683
      323.15  0.5494
      325.15  0.5315
      327.15  0.5146
      329.15  0.4985
      331.15  0.4832
      333.15  0.4688
      335.15  0.4550
      337.15  0.4418
      339.15  0.4293
      341.15  0.4174
      343.15  0.4061
      345.15  0.3952
      347.15  0.3849
      349.15  0.3750
      351.15  0.3655
      353.15  0.3565
      355.15  0.3478
      357.15  0.3395
      359.15  0.3315
      361.15  0.3239
      363.15  0.3165
      365.15  0.3095
      367.15  0.3027
      369.15  0.2962
      371.15  0.2899
      373.15  0.2838
   end
end
...
```

Description A look-up function is used to compute the values of the viscosity as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (`Water_Viscosity` in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the abscissa variable identified in the user-defined function (`T` in the example).

4.56.15 VISCOSITY = WELD

Parameters
```
[BETA = REAL]
C0 = REAL
C1 = REAL
C2 = REAL
C3 = REAL
T_LIQ = REAL T_90 = REAL T_MAX = REAL
```

Example
```
Viscosity = Weld C0=1 C1=-1e-2 C2=0 C3=0 T_LIQ=920 T_MAX=1400 T_90=1000
```

Description This is an empirical model that emulates the melting of a solid metal during the laser welding process.

$$\mu = \begin{cases} \mu_{90} + (\mu_{liq} - \mu_{90}) \frac{T - T_{90}}{T_{liq} - T_{90}} & : \quad T < T_{liq} \\ \\ c_0 + c_1 \hat{T} + c_2 \hat{T}^2 + c_3 \hat{T}^3 & : \quad T >= T_{liq} \end{cases} \tag{4.137}$$

where μ_{liq} is given by

$$\mu_{liq} = c_0 + c_1 T_{liq} + c_2 T_{liq}^2 + c_3 T_{liq}^3, \tag{4.138}$$

$\mu_{90} = \beta \mu_{liq}$ and $\hat{T} = \min(T, T_{max})$. The default value of `BETA` is 10^{11}.

4.57 FLOWING LIQUID VISCOSITY

FLOWING LIQUID VISCOSITY = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Specifies the material model for the fluid viscosity.

Summary Specifies the material model for the fluid viscosity in the Brinkman momentum equation.

Parent Block(s) `ARIA MATERIAL`

4.57.1 FLOWING LIQUID VISCOSITY = CONSTANT

Parameters `MU = REAL`

Example `FLOWING LIQUID VISCOSITY = CONSTANT MU = 1.0`

Description `MU` is the value of the constant fluid viscosity.

4.58 VOLUME FRACTION GAS

VOLUME FRACTION GAS = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Specifies the material model for the volume fraction of gas.

Summary Specifies the material model for the volume fraction of gas.

Parent Block(s) `ARIA MATERIAL`

4.58.1 VOLUME FRACTION GAS = CONSTANT

Parameters `value = REAL`

Example `Volume Fraction Gas = Constant value = 1.0`

Description `value` is the volume fraction of gas.

4.58.2 VOLUME FRACTION GAS = FROM_MASS_FRACTIONS

Parameters `None`

Example `Volume Fraction Gas = From_Mass_Fractions`

Description For a multiphase ChemEq model, this model uses

$$\phi = 1 - \rho \sum_s \frac{Y_i}{\rho_{b,i}} \tag{4.139}$$

where ϕ is the gas volume fraction, ρ is the overall density, $\rho_{b,i}$ is the bulk density of the i^{th} solid species, and Y_i is the mass fraction of the i^{th} solid species. This model uses the species phase definitions from the ChemEq definition to determine which species are solid. As such, the model can only be used in a material block which also contains a ChemEq definition.

This uses expressions for the mass fractions, so its material block must also include "Mass_Fraction of X = From_ChemEq" for each solid species, or some comparable definition for mass fraction.

You must also provide an expression for the bulk density of each solid species.

4.58.3 VOLUME FRACTION GAS = ENCORE_FUNCTION

Parameters

```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example

```
Volume Fraction Gas = Encore_Function Name=My_Function
```

Description

Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

4.58.4 VOLUME FRACTION GAS = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
Volume Fraction Gas= Polynomial Variable=Temperature Order=1 C0=401.0
C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$\kappa = \sum_{i=0}^{N} C_i \left(X + X_\circ\right)^i \tag{4.140}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

4.58.5 VOLUME FRACTION GAS = USER_FUNCTION

Parameters

```
NAME = STRING
X = STRING
```

Example

```
begin definition for function SI_phi
  type is piecewise linear
  begin values
     20.0     5.50e7
    100.0     4.60e7
    ...
    800.0     1.30e7
   2000.0     1.30e7
  end values
end definition for function SI_phi

...

Begin Aria Material Foo
   ...
   Volume Fraction Gas = User_Function Name=SI_phi X=Temperature
   ...
End Aria Material Foo
```

Description

A look-up function is used to compute the values of the thermal conductivity as a function of some other variable, i.e. $f(x)$. The function type ("piecewise linear" in the example above) must support the `differentiate()` method for Newton's method.

Here `NAME` is the name of the user-defined function (SI_phi, in the example) and `X` is the Aria name of the abscissa variable (`TEMPERATURE` in the example). Note that `X` is not necessarily the same name as the optional abscissa variable identified in the user-defined function.

4.59 YOUNGS MODULUS

YOUNGS MODULUS = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description

Specifies the material model for the Young's modulus.

Summary

The solid stress $\boldsymbol{T}$ is given by

$$\boldsymbol{T} = \lambda E_{kk}\boldsymbol{I} + 2\mu\boldsymbol{E} - \beta\left(T - T_{ref}\right)\boldsymbol{I} \tag{4.141}$$

where λ and μ are the Lamé coefficients, $\boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{\nabla d} + \boldsymbol{\nabla d}^T\right)$ is the deformation tensor, β is the coefficient of thermal stress, T is temperature and T_{ref} is the thermal strain reference temperature.

These Lamé coefficients are related to the more standard Young's modulus, Poisson's ratio

and CTE (α) as follows:

$$2\mu = \frac{E}{(1+\nu)} \tag{4.142}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu\frac{\nu}{(1-2\nu)} \tag{4.143}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha(3\lambda+2\mu) \tag{4.144}$$

When a user supplies the Young's modulus, Poisson's ratio and CTE properties Aria internally converts them into the Lamé coefficients.

Supplying the Lamé coefficients is more computationally efficient but perhaps less convenient, especially if the material properties are varying (e.g., temperature dependent in a non-isothermal problem).

Parent Block(s) `ARIA MATERIAL`

4.59.1 YOUNGS MODULUS = CONSTANT

Parameters `YM = REAL`

Example
```
YOUNGS MODULUS = CONSTANT YM = 1.0
```

Description `YM` is the value of the constant Young's modulus.

4.59.2 YOUNGS MODULUS = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
YOUNGS MODULUS = Encore_Function Name=My_Function
```

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

Chapter 5

Equations Aria Solves

5.1 Generalized Conservation Equation

We first introduce a general conservation equation, as a model for the specific equations that Aria solves, demonstrating how the Galerkin finite element method is applied to it, and how the integration by parts is carried out on its individual terms. Following [7], the conservation of a general scalar quantity $b(\boldsymbol{x}, t)$, with units of amount-per-unit-volume, at a point $\boldsymbol{x}$ and time t can be expressed as

$$\frac{\partial b}{\partial t} + \boldsymbol{\nabla} \cdot (b\boldsymbol{v}) = -\boldsymbol{\nabla} \cdot \boldsymbol{f} + B_{\mathrm{V}} \tag{5.1}$$

where $\boldsymbol{v}$ is the mass average velocity, $\boldsymbol{f}$ is the diffusive flux of b, and B_{V} is the volumetric source of b.

The Galerkin FEM (G/FEM) residual form of 5.1 is formed by bringing the right hand side terms to the left, multiplying by the FEM weight function ϕ^i and integrating over the volume V,

$$R_b^i = \int_{\mathrm{V}} \left(\frac{\partial b}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} b + b \boldsymbol{\nabla} \cdot \boldsymbol{v} + \boldsymbol{\nabla} \cdot \boldsymbol{f} - B_{\mathrm{V}} \right) \phi^i \, \mathrm{dV} = 0. \tag{5.2}$$

In many applications $\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$ so we ignore that term from here on. However, it is straight forward to account for this term via the source term B_{V}. Using the vector identity $(\boldsymbol{\nabla} \cdot \boldsymbol{f})\phi^i = \boldsymbol{\nabla} \cdot (\boldsymbol{f}\phi^i) - \boldsymbol{\nabla}\phi^i \cdot \boldsymbol{f}$ and using the divergence theorem, 5.2 becomes

$$R_b^i = \int_{\mathrm{V}} \left[\left(\frac{\partial b}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} b - B_{\mathrm{V}} \right) \phi^i - \boldsymbol{\nabla}\phi^i \cdot \boldsymbol{f} \right] \mathrm{dV} + \int_{\mathrm{S}} \boldsymbol{n} \cdot \boldsymbol{f} \phi^i \, \mathrm{dS} = 0. \tag{5.3}$$

Here $\boldsymbol{n}$ is a unit normal along the boundary S, pointing out of the volume V.

Equation 5.3 embodies the sign convention for sources, fluxes and equation terms used within Aria. For example, scalar flux expressions in Aria provide values for $f_n \equiv \boldsymbol{n} \cdot \boldsymbol{f}$ and should be positive for a flux of b leaving the volume V.

Note also that we have not assigned a units convention to the equation. Any unit system may be employed in the specification of the individual terms in 5.1. However, each term in 5.1 must have overall units of [b] / [time], and the overall residual expression has units of [b] * [L]**3 / [time], where [b] are units of the conserved quantity, b, [L] is the unit of the length scale, and [time] is the unit for time.

5.2 Conservation of Mass

For a material with density ρ, letting $b = \rho$ results in the conservation of mass. Since there is no net flow relative to the mass average velocity $\boldsymbol{f} = \boldsymbol{0}$. Although there are no sources of mass, having such a source

can be convenient in modeling and simulation; so, we let the mass source be $B_{\mathrm{V}} = q_m$. Thus, (5.1) becomes

$$\frac{\partial \rho}{\partial t} + \rho \boldsymbol{\nabla} \cdot \boldsymbol{v} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho = q_m. \tag{5.4}$$

For the special but common case of constant density, this reduces to

$$\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0. \tag{5.5}$$

Using equation 5.3, the G/FEM residual form is

$$R_P^i = \int_{\mathrm{V}} \left(-\frac{\partial \rho}{\partial t} - \rho \boldsymbol{\nabla} \cdot \boldsymbol{v} - \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho + q_m \right) \phi^i \, \mathrm{dV} = 0. \tag{5.6}$$

Important Note: Equation 5.6 has been multiplied by -1 because this form results in a better linear system for the special case of incompressible flow. This is important to remember when defining mass source terms.

In Aria, each term in 5.6 is specified separately as identified in equation 5.7.

$$R_P^i = \underbrace{\int_{\mathrm{V}} -\frac{\partial \rho}{\partial t} \phi^i \, \mathrm{dV}}_{\mathrm{MASS}} + \underbrace{\int_{\mathrm{V}} -\left(\boldsymbol{v} \cdot \boldsymbol{\nabla} \rho + \rho \boldsymbol{\nabla} \cdot \boldsymbol{v} \right) \phi^i \, \mathrm{dV}}_{\mathrm{ADV}} + \underbrace{\int_{\mathrm{V}} q_m \phi^i \, \mathrm{dV}}_{\mathrm{SRC}} = 0 \tag{5.7}$$

For a purely incompressible form, Aria offers the alternative form given in 5.8;

$$R_P^i + \underbrace{\int_{\mathrm{V}} -\boldsymbol{\nabla} \cdot \boldsymbol{v} \phi^i \, \mathrm{dV}}_{\mathrm{DIV}} + \underbrace{\int_{\mathrm{V}} q_m \phi^i \, \mathrm{dV}}_{\mathrm{SRC}} = 0 \tag{5.8}$$

5.3 Conservation of Energy

For a material with constant density and specific heat C_p, temperature T, heat flux $\boldsymbol{q}$ and volumetric energy source H_V, letting $b = \rho C_p T$, $\boldsymbol{f} = \boldsymbol{q}$ and $B_{\mathrm{V}} = H_V$ results in the conservation of energy.

$$\rho C_p \frac{\partial T}{\partial t} + \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T = -\boldsymbol{\nabla} \cdot \boldsymbol{q} + H_V. \tag{5.9}$$

A common constitutive relationship for $\boldsymbol{q}$ is Fourier's law, $\boldsymbol{q} = -\kappa \boldsymbol{\nabla} T$ where κ is the thermal conductivity. However, we leave the heat flux as an option to be specified as part of the material properties (see section 4.15). Using equation 5.3, the G/FEM residual form is

$$R_T^i = \int_{\mathrm{V}} \left[\left(\rho C_p \frac{\partial T}{\partial t} + \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T - H_V \right) \phi^i - \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{q} \right] \mathrm{dV} + \int_{\mathrm{S}} q_n \phi^i \, \mathrm{dS} = 0 \tag{5.10}$$

where q_n is the heat flux at the boundary. For example, the natural convection boundary condition gives $q_n = h(T - T_\infty)$ where h is the heat transfer coefficient and T_∞ is the bulk temperature away from the surface.

In Aria, each term in 5.10 is specified separately as identified in equation 5.11.

$$R_T^i = \underbrace{\int_{\mathrm{V}} \rho C_p \frac{\partial T}{\partial t} \phi^i \, \mathrm{dV}}_{\mathrm{MASS}} + \underbrace{\int_{\mathrm{V}} \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T \phi^i \, \mathrm{dV}}_{\mathrm{ADV}} - \underbrace{\int_{\mathrm{V}} H_V \phi^i \, \mathrm{dV}}_{\mathrm{SRC}}$$
$$- \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{q} \, \mathrm{dV}}_{\mathrm{DIFF}} + \int_{\mathrm{S}} q_n \phi^i \, \mathrm{dS} = 0 \tag{5.11}$$

More and more often we needing to account for variable density problems and so we need to bring back some of the terms we threw away because we were going to assume $\boldsymbol{\nabla} \cdot \boldsymbol{v} \equiv 0$. Here's a do-over of equation 5.11 that accommodates a variable density through the DIV term:

$$R_T^i = \underbrace{\int_{\mathrm{V}} \rho C_p \frac{\partial T}{\partial t} \phi^i \, \mathrm{dV}}_{\mathrm{MASS}} + \underbrace{\int_{\mathrm{V}} \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T \phi^i \, \mathrm{dV}}_{\mathrm{ADV}} + \underbrace{\int_{\mathrm{V}} \rho C_p T \boldsymbol{\nabla} \cdot \boldsymbol{v} \phi^i \, \mathrm{dV}}_{\mathrm{DIV}} - \underbrace{\int_{\mathrm{V}} H_V \phi^i \, \mathrm{dV}}_{\mathrm{SRC}} - \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{q} \, \mathrm{dV}}_{\mathrm{DIFF}} + \int_{\mathrm{S}} q_n \phi^i \, \mathrm{dS} = 0 \quad (5.12)$$

Note, however, that equation 5.12 still assumes a constant specific heat C_p.

5.3.1 SUPG

SUPG stabilization can be activated by added the `SUPG` term on the input file line. This capability is still under development.

5.4 Conservation of Chemical Species

For a material with species k with molar concentration C_k, molar flux $\boldsymbol{J}_k$ relative to the mass average velocity and volumetric reaction rate $R_{V,k}$, letting $b = y_k$, $\boldsymbol{f} = \boldsymbol{J}_k$ and $B_{\mathrm{V}} = R_{V,k}$ in (5.1) results in the conservation equation for species k,

$$\frac{\partial C_k}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} C_k = -\boldsymbol{\nabla} \cdot \boldsymbol{J}_k + R_{V,k}. \quad (5.13)$$

For liquid mixtures which are dilute in all species except one, Fick's law is often used to approximate $\boldsymbol{J}_k$. In this approximation, D_k represents the diffusion coefficient of species k with respect to the concentrated species and it is assumed that the interactions between dilute species is assumed negligible. Again, however, we choose to leave the governing equation in the more general form and require the particular diffusive flux model as user input (see section 4.46). Using equation 5.3, the G/FEM residual form is

$$R_{C_k}^i = \int_{\mathrm{V}} \left[\left(\frac{\partial C_k}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} C_k - R_{V,k} \right) \phi^i - \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{J}_k \right] \mathrm{dV} + \int_{\mathrm{S}} q_{n,k} \phi^i \, \mathrm{dS} = 0 \quad (5.14)$$

where $q_{n,k}$ is the mass flux at the boundary. For example, the natural convection boundary condition gives $q_n = k(C_k - C_{\infty,k})$ where k is the mass transfer coefficient and $C_{\infty,k}$ is the bulk concentration away from the surface.

In Aria, each term in 5.14 is specified separately as identified in equation 5.15.

$$R_{C_k}^i = \underbrace{\int_{\mathrm{V}} \frac{\partial C_k}{\partial t} \phi^i \, \mathrm{dV}}_{\mathrm{MASS}} + \underbrace{\int_{\mathrm{V}} \boldsymbol{v} \cdot \boldsymbol{\nabla} C_k \phi^i \, \mathrm{dV}}_{\mathrm{ADV}} - \underbrace{\int_{\mathrm{V}} R_{V,k} \phi^i \, \mathrm{dV}}_{\mathrm{SRC}} - \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{J}_k \, \mathrm{dV}}_{\mathrm{DIFF}} + \int_{\mathrm{S}} q_{n,k} \phi^i \, \mathrm{dS} = 0 \quad (5.15)$$

More and more often we need to account for variable density problems and so we need to bring back some of the terms we threw away because we were going to assume $\boldsymbol{\nabla} \cdot \boldsymbol{v} \equiv 0$. Here's a do-over of equation 5.15 that accommodates a variable density through the DIV term:

$$R^i_{C_k} = \underbrace{\int_V \frac{\partial C_k}{\partial t}\phi^i \,\mathrm{dV}}_{\text{MASS}} + \underbrace{\int_V \boldsymbol{v}\cdot\boldsymbol{\nabla} C_k\phi^i \,\mathrm{dV}}_{\text{ADV}} + \underbrace{\int_V C\boldsymbol{\nabla}\cdot\boldsymbol{v}\phi^i \,\mathrm{dV}}_{\text{DIV}} - \underbrace{\int_V R_{V,k}\phi^i \,\mathrm{dV}}_{\text{SRC}} - \underbrace{\int_V \boldsymbol{\nabla}\phi^i\cdot\boldsymbol{J}_k \,\mathrm{dV}}_{\text{DIFF}} + \int_S q_{n,k}\phi^i \,\mathrm{dS} = 0 \quad (5.16)$$

Often times it is useful to solve for mass, weight or volume fractions of each species rather than for the concentration directly. In that case, an additional condition exists,

$$\sum_k C_k = 1 \quad (5.17)$$

Using this condition, it is only necessary to solve for $N-1$ species fractions where N is the total number of species present in the problem. The final species, then, is simply given as

$$C_j = 1 - \sum_{k\neq j} C_k \quad (5.18)$$

This method can be triggered in Aria by specifying the equation term FRACBAL. In this case, the equation for C_j is not included in the system of unknowns but is instead post-processed on the fly. Aria will automatically detect all other species equations and include them in the fraction balance.

5.5 Conservation of Fluid Momentum

The Cauchy momentum equation is given by

$$\rho\frac{\partial \boldsymbol{v}}{\partial t} + \rho\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v} - \boldsymbol{g} - \boldsymbol{\nabla}\cdot\boldsymbol{T} = \boldsymbol{0} \quad (5.19)$$

where $\boldsymbol{T}$ is the fluid stress tensor and $\boldsymbol{g}$ is a body force. We construct the G/FEM residual form of 5.19 by contracting with the unit coordinate vector in the k-direction, $\boldsymbol{e}_k$, multiplying by the weight function ϕ^i and integrating over the volume. Using the vector identity $(\boldsymbol{\nabla}\cdot\boldsymbol{T})\cdot\boldsymbol{e}_k\phi^i = \boldsymbol{\nabla}\cdot(\boldsymbol{T}\cdot\boldsymbol{e}_k\phi^i) - \boldsymbol{T}^t : \boldsymbol{\nabla}(\boldsymbol{e}_k\phi^i)$ and integrating by parts gives

$$R^i_{m,k} = \int_V \left[\left(\rho\frac{\partial \boldsymbol{v}}{\partial t} + \rho\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v} - \boldsymbol{g}\right)\cdot\boldsymbol{e}_k\phi^i + \boldsymbol{T}^t : \boldsymbol{\nabla}\left(\boldsymbol{e}_k\phi^i\right)\right] \mathrm{dV} - \int_S \boldsymbol{n}\cdot\boldsymbol{T}\cdot\boldsymbol{e}_k\phi^i \,\mathrm{dS} = 0 \quad (5.20)$$

In Aria, each term in 5.20 is specified separately as identified in equation 5.15.

$$R^i_{m,k} = \underbrace{\int_V \rho\frac{\partial \boldsymbol{v}}{\partial t}\cdot\boldsymbol{e}_k\phi^i \,\mathrm{dV}}_{\text{MASS}} + \underbrace{\int_V \rho\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}\cdot\boldsymbol{e}_k\phi^i \,\mathrm{dV}}_{\text{ADV}} - \underbrace{\int_V \boldsymbol{g}\cdot\boldsymbol{e}_k\phi^i \,\mathrm{dV}}_{\text{SRC}} + \underbrace{\int_V \boldsymbol{T}^t : \boldsymbol{\nabla}\left(\boldsymbol{e}_k\phi^i\right) \,\mathrm{dV}}_{\text{DIFF}} - \int_S \boldsymbol{n}\cdot\boldsymbol{T}\cdot\boldsymbol{e}_k\phi^i \,\mathrm{dS} = 0 \quad (5.21)$$

5.6 Conservation of Solid Momentum

Aria currently solves the quasistatic form of the solid momentum equations. Furthermore, the solid stress is treated as a linear elastic material. In this limit, the Cauchy momentum equation is given by

$$\boldsymbol{\nabla} \cdot \boldsymbol{T} = \mathbf{0} \tag{5.22}$$

where $\boldsymbol{T}$ is the solid stress tensor. We construct the G/FEM residual form of 5.19 by contracting with the unit coordinate vector in the k-direction, $\boldsymbol{e}_k$, multiplying by the weight function ϕ^i and integrating over the volume. Using the vector identity $(\boldsymbol{\nabla} \cdot \boldsymbol{T}) \cdot \boldsymbol{e}_k \phi^i = \boldsymbol{\nabla} \cdot (\boldsymbol{T} \cdot \boldsymbol{e}_k \phi^i) - \boldsymbol{T}^t : \boldsymbol{\nabla}(\boldsymbol{e}_k \phi^i)$ and integrating by parts gives

$$R^i_{m,k} = \int_{\mathrm{V}} \boldsymbol{T}^t : \boldsymbol{\nabla} \left(\boldsymbol{e}_k \phi^i \right) \, \mathrm{dV} = 0 \tag{5.23}$$

Here, the surface contribution, $\int_{\mathrm{S}} \boldsymbol{n} \cdot \boldsymbol{T} \cdot \boldsymbol{e}_k \phi^i \, \mathrm{dS}$, has been dropped because Aria currently only supports Dirichlet and natural (homogeneous) boundary conditions for the solid equation.

In Aria, each term in 5.23 is specified separately as identified in equation 5.24.

$$R^i_{m,k} = \underbrace{\int_{\mathrm{V}} \boldsymbol{T}^t : \boldsymbol{\nabla} \left(\boldsymbol{e}_k \phi^i \right) \, \mathrm{dV}}_{\mathrm{DIFF}} = 0 \tag{5.24}$$

Currently, Aria does not support direct specification of the more popular stress-strain parameterization that utilizes Young's modulus E, Poisson's ratio ν and coefficient of thermal expansion α (note, the shear modulus $G = \mu$). The relationship between these two parametizations is summarized here for convenience.

$$2\mu = \frac{E}{(1+\nu)} \tag{5.25}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu \frac{\nu}{(1-2\nu)} \tag{5.26}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha (3\lambda + 2\mu) \tag{5.27}$$

When the solid momentum equation is solved, the physical coordinates of the region, used to calculate integration quantities, is deformed by the displacement vector. The current configuration, $\mathbf{x}$, is calculated from the reference configuration, $\mathbf{X}$, by

$$\mathbf{x} = \mathbf{X} + \mathbf{d}, \tag{5.28}$$

where $\mathbf{d}$ is the displacement field. The displacement field is chosen automatically by the code, but may be overridden by setting a solution option. These displaced, physical coordinates are used for integration of all of the equations.

Additionally, the code supports the integration of the solid momentum equations in the reference configuration, rather than in the current configuration. This is activated through the use of the REF_DIFF equation term rather than the DIFF term. When this is used, the Cauchy stress tensor is rotated into the Second Piola-Kirchhoff stress tensor. Integration quantities are also calculated in the reference configuration. Care should be taken to ensure that the constitutive model used also changes its strain measure to one which is work conjugate with the Second Piola-Kirchhoff stress tensor.

If both the MESH and SOLID equations are defined in the same region, then the TALE mesh motion algorithm is activated automatically. Here, the reference frame for the SOLID equation is changed to the TALE reference frame, which is defined as

$$\underline{\mathbf{x}}_r = \underline{\mathbf{X}} + \underline{\mathbf{d}}_m - \underline{\mathbf{d}}_s \tag{5.29}$$

where $\underline{\mathbf{x}}_r$ is the new TALE reference frame, $\underline{\mathbf{d}}_m$ is MESH_DISPLACEMENTS, and $\underline{\mathbf{d}}_s$ is SOLID_DISPLACEMENTS. This affects the calculation of strain measures, such as $\underline{\underline{\mathrm{F}}}$, and integration quantities, such as $\det(J)$, among others. More details of the TALE method will be available soon in a pending report.

5.7 Voltage Equation

The electric potential or voltage V is frequently used in determining the electric field, $\boldsymbol{E} = -\boldsymbol{\nabla} V$. While (5.1) cannot be applied to the voltage, the equation governing the voltage – Gauss' law from Maxwell's equations – has a similar form. Writing the electric displacement $\boldsymbol{D}$ as $\boldsymbol{D} = \epsilon \boldsymbol{E}$, where ϵ is the electric permittivity, Gauss' law is

$$-\boldsymbol{\nabla} \cdot \epsilon \boldsymbol{\nabla} V = \rho_e \tag{5.30}$$

where the permittivity is taken to be a constant and ρ_e is the volumetric free charge density.

Using equation 5.3, the G/FEM residual form is

$$R_V^i = \int_{\mathrm{V}} \left(-\rho_e \phi^i + \boldsymbol{\nabla} \phi^i \cdot \epsilon \boldsymbol{\nabla} V \right) \,\mathrm{dV} + \int_{\mathrm{S}} q_n \phi^i \,\mathrm{dS} = 0 \tag{5.31}$$

In Aria, each term in 5.31 is specified separately as identified in equation 5.32.

$$R_V^i = \underbrace{-\int_{\mathrm{V}} \rho_e \phi^i \,\mathrm{dV}}_{\mathrm{SRC}} + \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla} \phi^i \cdot \epsilon \boldsymbol{\nabla} V \,\mathrm{dV}}_{\mathrm{DIFF}} + \int_{\mathrm{S}} q_n \phi^i \,\mathrm{dS} = 0 \tag{5.32}$$

5.8 Current Equation

An alternate formulation for solving for the electrical potential (see section 5.7) is to solve the "current" equation which is a conservation equation for electrical charge. The electrical current $\boldsymbol{J}$ is frequently related to the electric field $\boldsymbol{E}$ using Ohm's law as $\boldsymbol{J} = \sigma_e \boldsymbol{E}$ where σ_e is the electrical conductivity. The electric potential or voltage V is used in determining the electric field, $\boldsymbol{E} = -\boldsymbol{\nabla} V$. However, we choose to leave the electrical current as a more general constitutive model to be provided as a material model input (see section 4.6). More complex version for charged ion fluxes (Nerst Planck, etc. can be utilized with the species equation).

$$-\boldsymbol{\nabla} \cdot \boldsymbol{J} = S \tag{5.33}$$

Here S is a current source term, for example from electrochemical reactions in a battery simulation.

Using equation 5.3, the G/FEM residual form is

$$R_V^i = \int_{\mathrm{V}} \left(-S \phi^i - \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{J} \right) \,\mathrm{dV} + \int_{\mathrm{S}} q_n \phi^i \,\mathrm{dS} = 0 \tag{5.34}$$

In Aria, each term in 5.34 is specified separately as identified in equation 5.35.

$$R_V^i = \underbrace{-\int_{\mathrm{V}} S \phi^i \,\mathrm{dV}}_{\mathrm{SRC}} - \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{J} \,\mathrm{dV}}_{\mathrm{DIFF}} + \int_{\mathrm{S}} q_n \phi^i \,\mathrm{dS} = 0 \tag{5.35}$$

5.9 Charge Density Equation

This equation solves for the volumetric free charge density, ρ_e, and is meant to be coupled with the voltage equation in section 5.7). This equation can also be used in place of the current equation 5.33, since this equation has the correct form (the current equation has the correct form with ρ_e when electric displacement is activated) and the charge density is treated as a variable. The charge density evolves as

$$\frac{\partial \rho_e}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho_e = -\boldsymbol{\nabla} \cdot \boldsymbol{J} = \boldsymbol{\nabla} \cdot (\sigma \boldsymbol{\nabla} V) \tag{5.36}$$

where $\boldsymbol{J} = -\sigma \boldsymbol{\nabla} V$ is assumed to obey Ohm's law and σ represents the conductivity.

The G/FEM residual form of this equation is

$$R_V^i = \int_{\mathrm{V}} \left(\frac{\partial \rho_e}{\partial t} \phi^i + \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{v} + \boldsymbol{\nabla} \phi^i \cdot \sigma \boldsymbol{\nabla} V \right) \mathrm{dV} + \int_{\mathrm{S}} q_n \phi^i \, \mathrm{dS} = 0 \tag{5.37}$$

In Aria, each term in 5.37 is specified separately as identified in equation 5.38.

$$R_V^i = \underbrace{\int_{\mathrm{V}} \frac{\partial \rho_e}{\partial t} \phi^i \, \mathrm{dV}}_{\mathrm{MASS}} + \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla} \phi^i \cdot \boldsymbol{v} \, \mathrm{dV}}_{\mathrm{ADV}} + \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla} \phi^i \cdot \sigma \boldsymbol{\nabla} V \, \mathrm{dV}}_{\mathrm{DIFF}} + \int_{\mathrm{S}} q_n \phi^i \, \mathrm{dS} = 0. \tag{5.38}$$

5.10 Suspension Equation

In treating the suspension as a continuum, we introduce an evolution equation for particle volume fraction, ϕ, as

$$\frac{\partial \phi}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \phi + \boldsymbol{\nabla} \cdot \boldsymbol{N} = 0. \tag{5.39}$$

The particle volume fraction is defined as the total summed volume of particles per volume of the particle medium. 5.39 represents a balance between the stored particles, the convected particle flux, and the diffusive particle flux, N. Several mechanisms which include Brownian motion, sedimentation, hydrodynamic particle interactions, and gradients in suspension viscosity may contribute to the diffusive particle flux. Specification of the appropriate flux model must then be carried out to close the definition of the conservation equation.

5.11 Porous Flow Equations

In what follows we use a subscript α to denote a component index and N_α is the total number of components. We use a subscript β to denote a phase and the total number of phases is N_β. The subscript s is used to refer to the solid phase of the porous skeleton; this phase is not included in N_β. Summations over α and β are assumed to range from one to N_α and N_β, respectively, unless otherwise specified.

5.11.1 Generalized Equations

This write up follows, directly, the reports of [8], [9] and [10].

Porosity and Void Ratio

In modeling of porous materials, one often needs to know the relative volume of pore space (V_p) and the solid porous skeleton (V_s). The most common measure of this is the *porosity*, ϕ, which is defined as the ratio of the volume of pore space (V_p) per unit total volume ($V_t = V_p + V_s$), viz.,

$$\phi = \frac{V_p}{V_t} = \frac{V_p}{V_p + V_s}. \tag{5.40}$$

The *void ratio* (r) is sometimes used as well and is defined as the ratio of the pore volume to the solid volume, viz.,

$$r = \frac{V_p}{V_s} \tag{5.41}$$

Clearly the porosity and void ratio are related,

$$r = \frac{\phi}{1-\phi} \tag{5.42}$$

$$\phi = \frac{r}{1+r} \tag{5.43}$$

Component Mass Balances

The mass balance for each component across all phases can be written

$$\frac{\partial d_\alpha}{\partial t} + \boldsymbol{\nabla} \cdot \boldsymbol{F}_\alpha = Q_\alpha \qquad \alpha = 1, \ldots, N_\alpha \tag{5.44}$$

where d_α is mass concentration of component α across all phases, t is time, $\boldsymbol{F}_\alpha$ is the total mass flux of component α relative to a fixed frame of reference and Q_α is the volumetric source of component α.

The mass concentration of component α can be written as

$$d_\alpha = \phi \sum_\beta Y_{\alpha\beta} \rho_\beta S_\beta \tag{5.45}$$

where ϕ is the interstitial volume fraction, a.k.a. the porosity, that ranges from zero to one, Y denotes mass fraction so that $Y_{\alpha\beta}$ is the mass fraction of component α in phase β, ρ_β is mixture density of phase β and S_β is saturation which is defined as the fraction of the interstitial pore space occupied by phase β.

For completness it is noted here that the mass fractions sum to one in each phase and that the saturations of all phases sum to unity,

$$\sum_\alpha Y_{\alpha\beta} = 1 \tag{5.46}$$

$$\sum_\beta S_\beta = 1. \tag{5.47}$$

The macroscopic mass flux in phase β, $\rho_\beta \boldsymbol{v}_\beta$ is obtained from the extended Darcy's Law,

$$\rho_\beta \boldsymbol{v}_\beta = \boldsymbol{f}_\beta = -\frac{\rho_\beta \kappa_{r\beta}}{\mu_\beta} \boldsymbol{K} \cdot (\boldsymbol{\nabla} P_\beta + \rho_\beta \boldsymbol{g}) \tag{5.48}$$

where $\kappa_{r\beta}$ is the relative permeability of phase β and $\boldsymbol{K}$ is the permeability tensor which are both intrinsic properties of the porous material and so may vary spatialy. The mixture viscosity of phase β is μ_β, P_β is the pressure in phase β and $\boldsymbol{g}$ is the gravitational acceleration.

The component mass flux, $\boldsymbol{F}_\alpha$ can be written as the sum of the component mass fluxes across each phase,

$$\boldsymbol{F}_\alpha = \sum_\beta \boldsymbol{F}_{\alpha\beta}. \tag{5.49}$$

Within each phase, the component mass flux can be decomposed in its convective and diffusive components,

$$\boldsymbol{F}_{\alpha\beta} = Y_{\alpha\beta}\boldsymbol{f}_\beta + \boldsymbol{J}_{\alpha\beta} \tag{5.50}$$

where $\boldsymbol{J}_{\alpha\beta}$ is the diffusive flux of component α in phase β. For example, for a gaseous, dilute binary mixture the diffusive flux may be modeled using Fick's Law,

$$\boldsymbol{J}_{\alpha\beta} = -\rho_\beta \mathcal{D}_{\alpha\beta}\boldsymbol{\nabla} Y_{\alpha\beta} \qquad \text{(dilute binary gas mixture)} \tag{5.51}$$

where $\mathcal{D}_{\alpha\beta}$ is the diffusion cofficient of component α in phase β.

Porous Species

The porous species equation is identical to the mass balance equation, with all quantities being molar concentrations. The porous species equation is written as

$$\frac{\partial c_\alpha}{\partial t} + \boldsymbol{\nabla}\cdot\boldsymbol{\Pi}_\alpha = R_\alpha \qquad \alpha = 1,\ldots,N_\alpha, \tag{5.52}$$

where c_α is the molar concentration, defined analogously to (5.45), Π_α is the molar flux, and R_α is the volumetric molar source of component α. The porous species equation exists to facilitate using expressions derived for use with the non porous-flow species equation. An analogous set of terms to those found in the mass balance equation can be derived on a molar concentration basis.

Energy Balance

In order to accomodate multicomponent transport problems we employ a enthalpy based energy conservation law,

$$\frac{\partial e}{\partial t} + \boldsymbol{\nabla}\cdot\boldsymbol{q} = Q_e \tag{5.53}$$

where e is the total internal energy, $\boldsymbol{q}$ is the total heat flux relative to a stationary frame of reference and Q_e is the volumetric source of energy. The total internal energy is written as a sum of the energies of all components across all phases as well as the internal energy of the solid porous structure,

$$e = (1-\phi)\,\rho_s e_s + \phi\sum_\beta S_\beta \rho_\beta e_\beta \tag{5.54}$$

where ρ_β and e_β are the mixture density and internal energy of phase β. Generally, thermodynamic relations are required to relate these quantities to the component specific quantities.

The total heat flux can be decomposed into different modes of transport. Typically,

$$\boldsymbol{q} = -\lambda_T\boldsymbol{\nabla}T + \sum_\beta\sum_\alpha h_{\alpha\beta}\boldsymbol{F}_{\alpha\beta} \tag{5.55}$$

where λ_T is the effective thermal conductivity, T is the temperature and $h_{\alpha\beta}$ is the partial enthalpy of component α in phase β which is interpreted as the enthalpy of pure component α at the temperature and pressure of phase β. Using (5.50), we can expand (5.55) such that

$$\begin{aligned}\sum_\alpha h_{\alpha\beta}\boldsymbol{F}_{\alpha\beta} &= \sum_\alpha h_{\alpha\beta}\left(Y_{\alpha\beta}\boldsymbol{f}_\beta + \boldsymbol{J}_{\alpha\beta}\right)\\ &= h_\beta\boldsymbol{f}_\beta + \sum_\alpha h_{\alpha\beta}\boldsymbol{J}_{\alpha\beta}\end{aligned} \tag{5.56}$$

where $h_\beta = \sum_\alpha h_{\alpha\beta} Y_{\alpha\beta}$ is the mixture enthalpy of phase β. Thus, the total heat flux can be expressed as the sum of conductive, convective and diffusive contributions,

$$\boldsymbol{q} = -\lambda_T \boldsymbol{\nabla} T + \sum_\beta \rho_\beta \boldsymbol{v}_\beta h_\beta + \sum_\beta \sum_\alpha h_{\alpha\beta} \boldsymbol{J}_{\alpha\beta} \tag{5.57}$$

5.11.2 Porous Flow Systems

To facilitate various flavors of single phase and multiphase flow in Aria, the porous flow system model was created to manange expression creation. In addition to the single phase model (sections 5.11.2 and 5.11.2), there are several others available including: mixed single phase, CO2-salt-brine (section 5.11.2), two-phase-immisicible (sections 5.11.2 and 5.11.2), air-water (section 5.11.2), and organic-material-decomposition (section 5.11.2). The basic purpose of the porous flow system model is to create the necessary expressions for each system. There are examples of each flow system in the regression tests, which are the best place to start when developing an input for a similar system. The input syntax for a porous flow system is given in section 4.1 in addition to the required parameters for each porous flow system..

Single Phase, Single Component, Isothermal Flow

The simplest possible case is for a single phase flow ($N_\beta = 1$, $S = 1$) with a single component ($N_\alpha = 1$, $Y = 1$) at a fixed temperature. This is the single phase porous flow system. For a single component flow, there is no diffusive mass flux so $\boldsymbol{J} = \boldsymbol{0}$ and $\boldsymbol{F} = \boldsymbol{f}$ where

$$\boldsymbol{F} = \boldsymbol{f} = \rho \boldsymbol{v} = -\frac{\rho \kappa_r}{\mu} \boldsymbol{K} \cdot (\boldsymbol{\nabla} P + \rho \boldsymbol{g}) \tag{5.58}$$

In this case, the mass balance equation is all that is needed to solve for the unknown pressure field P.

$$\frac{\partial d}{\partial t} + \boldsymbol{\nabla} \cdot \boldsymbol{f} = Q \tag{5.59}$$

Since the density is most likely a function of temperature, the constant uniform temperature will have to be specified. Finally, in this special case, (5.45) reduces to $d = \phi \rho$.

Single Phase, Single Component, Non-isothermal Flow

Building on the simple case developed in section 5.11.2, the next step is to accomodate a non-isothermal flow. Here, we still have a single phase flow ($N_\beta = 1$, $S = 1$) with a single component ($N_\alpha = 1$, $Y = 1$). For a single component flow, there is no diffusive mass flux so $\boldsymbol{J} = \boldsymbol{0}$ and $\boldsymbol{F} = \boldsymbol{f}$ where

$$\boldsymbol{F} = \boldsymbol{f} = \rho \boldsymbol{v} = -\frac{\rho \kappa_r}{\mu} \boldsymbol{K} \cdot (\boldsymbol{\nabla} P + \rho \boldsymbol{g}) \tag{5.60}$$

Again, the mass balance equation is all that is needed to solve for the unknown pressure field P.

$$\frac{\partial d}{\partial t} + \boldsymbol{\nabla} \cdot \boldsymbol{f} = Q \tag{5.61}$$

Again, in this special case, (5.45) reduces to $d = \phi \rho$.

In order to solve for the unknown temperature, T, we solve an energy transport equation in the enthalpy form as described in section 5.11.1. Here, the energy flux in equation (5.55) reduces to

$$\boldsymbol{q} = -\lambda_T \boldsymbol{\nabla} T + \rho \boldsymbol{v} h \tag{5.62}$$

where h is the pure component enthalpy which may be a function of P and T.

Two-Phase Immiscible

In this section, we define a formulation for two-phase flow of immiscible fluids, that is, without mass transfer between the phases. Common examples include oil/water, under pressure and temperature conditions where phase transitions are negligible, and systems involving non- aqueous phase liquids (NAPL) occurring in subsurface contamination by organic liquids, e.g., solvents, fuels, oils. If we disallow partitioning of components across phases, then we get the following system of mass balance equations for a wetting and non-wetting phase,

$$\frac{\partial}{\partial t}(\rho_w \phi(1 - S_n)) = \boldsymbol{\nabla} \cdot \left(\rho_w \frac{k_{rw}}{\mu_w} \boldsymbol{K} \cdot (\boldsymbol{\nabla} p - \rho_w \boldsymbol{g}) \right) + Q_w \tag{5.63}$$

$$\frac{\partial}{\partial t}(\rho_n \phi S_n) = \boldsymbol{\nabla} \cdot \left(\rho_n \frac{k_{rn}}{\mu_n} \boldsymbol{K} \cdot (\boldsymbol{\nabla} p + \boldsymbol{\nabla} p_c - \rho_n \boldsymbol{g}) \right) + Q_n \tag{5.64}$$

In these equations, the subscript w represents wetting phase properties and the subscript n nonwetting phase properties. We have also incorporated the Darcy flux terms, including pressure and gravitational forces, as well as capillary pressure:

$$p_c = p_n - p_w \tag{5.65}$$

The symbol p in the equations is used in place of wetting phase pressure, pw. Also, the pore space is assumed saturated with fluids,

$$S_w + S_n = 1 \tag{5.66}$$

This constraint has been incorporated in the wetting phase mass balance. In addition to the density models described in the previous section, models for density and viscosity can be specified as constant, polynomial with respect to other primary or secondary variables, or by tabular functions, or a user-defined plug-in.

Both the capillary pressure and relative permeability are functions of phase saturations. Either of these can be specified by the user in a tabular form, as a function of either phase saturation. A plug-in can also be written by the user for either of these functions. Several commonly used functions are built-in to the code.

Pressure-pressure Formulation for Two-phase Immiscible

The $P_w - S_n$ formulation is generally known to have superior numerical behavior for $S_w \to 0$, however, the capillary pressure in heterogeneous media creates a problem for a vertex-centered (e.g. FEM) numerical scheme. In FEM, because a node straddles a material interface, which separates two materials with different texture and therefore different wetting properties, the capillary pressure is continuous across the interface, but the saturation is not. So, a (P_w,S_n) formulation in FEM would require dealing with discontinuous DOF for S_n. One alternative is to instead solve for two pressures. Two pairs of pressure-based DOFs available in Aria include (P_w, P_c) and (P_w, P_n). (Note that in single phase regions, where $S_w = 0$ or 1, then P_w or P_n are undefined. The formulation should work for the thermal battery problem so long as $S_w > 0$.

The following system of mass balance equations for a wetting and non-wetting phase describes an immiscible flow,

$$\frac{\partial}{\partial t}(\rho_w \phi S_w) = \boldsymbol{\nabla} \cdot \left(\rho_w \frac{k_{rw}}{\mu_w} \boldsymbol{K} \cdot (\boldsymbol{\nabla} P_w - \rho_w \boldsymbol{g}) \right) + Q_w \tag{5.67}$$

$$\frac{\partial}{\partial t}(\rho_n \phi S_n) = \boldsymbol{\nabla} \cdot \left(\rho_n \frac{k_{rn}}{\mu_n} \boldsymbol{K} \cdot (\boldsymbol{\nabla} P_w + \boldsymbol{\nabla} P_c - \rho_n \boldsymbol{g}) \right) + Q_n \tag{5.68}$$

In these equations, the subscript w represents wetting phase properties and the subscript n non-wetting phase properties. We have also incorporated the Darcy flux terms, including pressure and gravitational forces, as well as capillary pressure:

$$p_c = p_n - p_w \tag{5.69}$$

Also, the pore space is assumed saturated with fluids,

$$S_w + S_n = 1 \tag{5.70}$$

The current implementation in Aria uses a van Genuchten function to model the relationship between the capillary pressure and the saturation. This relationship is inverted for saturation given the capillary pressure in the pressure-pressure formulation.

Two-phase Air-Water

This model describes the nonisothermal two-component, two-phase transport of water and air in a porous medium. The air is treated as a non-condensible gas, but it can dissolve in the liquid phase. The water can exist as water vapor or liquid. The model allows the fluids to partition between the two phases, depending on pressure and temperature conditions. Two single phase states (all liquid or all gas) and one two-phase state are allowed.

The component mass balance equations describing two-phase transport of water, air and energy are:

$$\frac{\partial}{\partial t}(\phi(S_l Y_{wl}\rho_l + S_g Y_{wg}\rho_g)) + \boldsymbol{\nabla}\cdot(Y_{wl}\rho_l \boldsymbol{v}_l + Y_{wg}\rho_g \boldsymbol{v}_g + \boldsymbol{J}_{wg}) + Q_w = 0 \tag{5.71}$$

$$\frac{\partial}{\partial t}(\phi(S_l Y_{al}\rho_l + S_g Y_{ag}\rho_g)) + \boldsymbol{\nabla}\cdot(Y_{al}\rho_l \boldsymbol{v}_l + Y_{ag}\rho_g \boldsymbol{v}_g + \boldsymbol{J}_{ag}) + Q_a = 0 \tag{5.72}$$

$$\frac{\partial}{\partial t}\left[(1-\phi)\rho_s e_s + \phi(\rho_l S_l e_l + \rho_g S_g e_g)\right] + \boldsymbol{\nabla}\cdot(-\lambda_T \boldsymbol{\nabla} T + \rho_l \boldsymbol{v}_l h_l + \rho_g \boldsymbol{v}_g h_g + h_{wg}\boldsymbol{J}_{wg} + h_{ag}\boldsymbol{J}_{ag}) + Q_e = 0 \tag{5.73}$$

where

$$\boldsymbol{J}_{ag} = -\rho_g D_{wa} \boldsymbol{\nabla} Y_{ag} \tag{5.74}$$

is the gas-phase binary diffusion flux for water vapor through air. In these equations, subscript l refers to the liquid phase and g to the gas phase. In addition to previously defined variables: $Y_{\alpha\beta}$ is the mass fraction of component α (w for water, a for air) in phase β, S_β is the saturation of phase β, ρ_β is the phase density, e_β is the internal energy of phase β, λ_T is the effective thermal heat conduction, h_β is the enthalpy of phase β and $h_{\alpha g}$ is the component enthalpy in the gas phase.

The liquid and gas phase Darcy velocities are

$$\boldsymbol{v}_l = -\frac{k_{rl}}{\mu_l}\boldsymbol{K}\cdot(\boldsymbol{\nabla} P_l + \rho_l \boldsymbol{g}) \tag{5.75}$$

$$\boldsymbol{v}_g = -\frac{k_{rg}}{\mu_g}\boldsymbol{K}\cdot(\boldsymbol{\nabla} P_g + \rho_g \boldsymbol{g}) \tag{5.76}$$

In addition, the following constraints hold:

$$S_l + S_g = 1\,; \quad Y_{w\beta} + Y_{a\beta} = 1\,, \text{for } \beta = l \text{ or } g \tag{5.77}$$

CO2-salt-brine

The CO2-salt-brine system is intended to model the interaction of CO2 gas, solid salt, and liquid brine. In this system there are three mass balance equations that must be solved for the saturations of each. This formulation describes a 3-component 2-phase flow for the water-CO2-NaCl system. The two phases are brine, denoted by subscript w, a liquid phase of water with salt and CO2 dissolved in it, and "gas" denoted by subscript g, a CO2-rich phase, with some water dissolved in it, but no salt. The formulation also deals with precipitation of salt from the liquid phase.

To deal with the precipitation of NaCl from the liquid to form solid salt, the fluid system is treated as a three-phase system, with two flowing phases composed of the brine and CO2 phases, and one non-flowing solid phase, which is the solid salt precipitate. Obviously, the formulation holds for other chloride salts, e.g. CaCl, KCl.

$$\frac{\partial}{\partial t}\phi(S_w\rho_w x_w + S_g\rho_g y_w) + \boldsymbol{\nabla}\cdot(\rho_w x_w \boldsymbol{v}_w + \rho_g y_w \boldsymbol{v}_g + \boldsymbol{J}_w^{H2O} + \boldsymbol{J}_g^{H2O}) = Q^{H2O} \tag{5.78}$$

$$\frac{\partial}{\partial t}\phi(S_w\rho_w x_{CO2} + S_g\rho_g y_{CO2}) + \boldsymbol{\nabla}\cdot(\rho_w x_{CO2} \boldsymbol{v}_w + \rho_g y_{CO2} \boldsymbol{v}_g + \boldsymbol{J}_w^{CO2} + \boldsymbol{J}_g^{CO2} = Q^{CO2} \tag{5.79}$$

$$\frac{\partial}{\partial t}\phi(S_w\rho_w x_{NaCl} + S_g\rho_s) + \boldsymbol{\nabla}\cdot(\rho_w x_{NaCl} \boldsymbol{v}_w + \boldsymbol{J}_w^{NaCl}) = Q^{NaCl} \tag{5.80}$$

In the equations above, we have used the following nomenclature, S_s is the saturation of solid salt precipitate, S_w is the brine phase saturation, S_g is the CO2-rich phase saturation, x_w is the mass fraction of water in the brine phase, x_{CO2} is the mass fraction of CO2 in the brine phase, x_{NaCl} is the mass fraction of salt dissolved in the brine phase, y_{CO2} is the mass fraction of CO2 in the CO2-rich ("gas") phase, y_w is the mass fraction of water in the gas phase, $\rho_s(T,p)$ is the density of solid precipitate, and $\boldsymbol{J}_\beta^\alpha$ is the diffusion flux of component α in phase β which can also include dispersion. The subscript 'w' refers to the brine phase, 'g' refers to the CO2 phase and 's' refers to the solid salt phase.

This model treats all the separate-phase CO2 as a single CO2-rich phase. The phase diagram for the CO2-H20 system displays a saturation line where liquid water, liquid CO2 and a vapor phase can co-exist. The saturated water vapor pressure on this line is small, so the vapor phase is mostly CO2.

$$S_w + S_g + S_s = 1\,; \quad x_w + x_{CO2} + X_{NaCl} = 1\,; \quad y_w + y_{CO2} + y_{NaCl} = 1 \tag{5.81}$$

We also have the constraint on the over-all composition, which can replace either of the last two constraints above,

$$z_w + z_{CO2} + z_{NaCl} = 1 \tag{5.82}$$

To solve for the pressure, in all compositional situations, we can solve a pressure equation, obtained by summing all component balances. Since the resulting equation is not independent, it replaces one of the three mass balance equations presented earlier.

$$\frac{\partial}{\partial t}\phi(S_w\rho_w + S_g\rho_g + S_s\rho_s) + \boldsymbol{\nabla}\cdot(\rho_w \boldsymbol{v}_w + \rho_g \boldsymbol{v}_g) + Q^{H2O} + Q^{CO2} + Q^{NaCl} = 0 \tag{5.83}$$

In the numerical implementation, this equation could be obtained after all BCs have been applied to the discrete equations, by summing the three residual equations together.

Recall, the primary variables for this system are defined as a set of variables that allow one to compute all the remaining (secondary) variables appearing in the mass and energy balance equations. For this problem, the primary solution vector (assuming we will also solve an energy balance for T) is $[P, z_{CO2}, \rho_{NaCl}, T]$, where

$$z_{CO2} = \frac{S_w\rho_w x_{CO2} + S_g\rho_g y_{CO2}}{S_w\rho_w + S_g\rho_g} = (1-\nu)x_{CO2} + \nu y_{CO2} \tag{5.84}$$

is the over-all mass fraction of CO2 in the flowing phases. We have introduced the fraction of gas, defined as,

$$\nu = \frac{\rho_g S_g}{\rho_w S_w + \rho_g S_g} \tag{5.85}$$

Similarly, the overall density of salt is defined by,

$$\rho_{NaCl} = \rho_w S_w x_{NaCl} + \rho_s S_s = (1-\nu) x_{NaCl} F + \rho_s S_s \tag{5.86}$$

where we have introduced, for later convenience, the following mass density,

$$F = \rho_w S_w + \rho_g S_g \tag{5.87}$$

Note that the accumulation term in the CO2 balance can be written as $\phi F z_{CO2}$.

In this formulation, P is solved for from the pressure equation, z_{CO2} from the CO2 mass balance equation, ρ_{NaCl} from the NaCl mass balance and T from the energy balance equation. We assume the phase equilibrium for this system is done ahead of time.

The thermodyanmics algorithm for this system is still in development and has not yet been completed.

Organic Material Decomposition (OMD)

Organic material decomposition (OMD) models currently define two-phase (solid and gas) problems in which the solid (or porous) species react to produce additional gas species that pressurize a system and volumetric heat transfer between phases. The notation used above does not include the porous solid as a phase, but since for OMD we are solving equations for the solid phase including reactions, we set $N_\beta = 2$, with it being understood that one of these phases is the solid phase. OMD problems will extend to include a liquid phase in the future. The OMD equations described here are based on the combustion model [11]. Further details about coupling boundary conditions between fluid-only and porous regions are provided in section 9.5. There are separate mass balance equations for the solid phase and gas phase. The solid phase mass balance equation is used to solve for solid phase density:

$$\underbrace{\frac{\partial \bar{\rho}}{\partial t}}_{MASS} = \underbrace{-\dot{\omega}'''_{f,g}}_{SRC}. \tag{5.88}$$

where $\bar{\rho}$ is the solid phase density and $\dot{\omega}'''_{f,g}$ is the rate of formation of gases due to heterogenous reaction kinetics. The mass balance equation for the gas phase solves for the pressure (from the gas phase density via the ideal gas law),

$$\underbrace{\frac{\partial(\psi\rho_g)}{\partial t}}_{MASS} + \underbrace{\frac{\partial(\rho_g u_{j,g})}{\partial x_j}}_{ADV} = \underbrace{\dot{\omega}'''_{f,g}}_{SRC}, \tag{5.89}$$

where ρ_g is the gas-phase density and ψ is the mixture-averaged solid-phase porosity. The porosity is the void space not taken up by the volume fractions of each solid phase species, typically defined as a function of the volume fractions of each solid species,

$$\psi = 1 - \sum_\alpha X_\alpha. \tag{5.90}$$

The volume fraction of each solid species is calculated as

$$X_\alpha = \frac{\bar{\rho} Y_\alpha}{\rho_{s0,\alpha}}, \tag{5.91}$$

where $\rho_{s0,\alpha}$ is the solid (non-porous) density for each species. In equation 5.89, $u_{j,g}$ is the gas-phase velocity and gets computed from Darcy's approximation as

$$\psi v_{j,g} = u_{j,g} = -\frac{\bar{K_{ij}}}{\mu_g}\left(\frac{\partial p_g}{\partial x_i} + \rho_g g_i\right), \tag{5.92}$$

where $\bar{K_{ij}}$ is the mixture-averaged solid-phase permeability tensor defined as the normalized volume average of the solid phase species' permeabilities,

$$\bar{K_{ij}} = \frac{\sum_\alpha X_\alpha K_{\alpha j}}{\sum_\alpha X_\alpha} \tag{5.93}$$

where

$$K_{\alpha j} = K_{ref}\left(\frac{T}{T_{ref}}\right)^n \delta_{\alpha j} \tag{5.94}$$

The intrinsic permeability is the product of the intrinsic permeability scaling and the intrinsic permeability. The reason it is separated in this way is that scalar models have access to many generic functions, whereas tensor entries can only be constants. A more complex function for the intrinsic permeability tensor would have to be written as a user subroutine.

In equation 5.92 the parameter μ_g is the gas-phase viscosity and g_j is the gravity vector. The ideal gas law gets used for the gas density in the pressure form of equation 5.89

$$\rho_g = \frac{P\bar{M}}{RT_g}. \tag{5.95}$$

Therefore the actual equation for pressure being solved here is

$$\frac{\partial}{\partial t}\left(\frac{p_g \bar{M}\psi}{RT_g}\right) + \frac{\partial}{\partial x_j}\left[\frac{p_g\bar{M}}{RT_g}\frac{\bar{K_{\alpha j}}}{\mu_g}\left(\frac{\partial p_g}{\partial x_j} + \frac{p_g\bar{M}}{RT_g}g_j\right)\right] = 0. \tag{5.96}$$

where $\bar{M}$ is the mass averaged molecular weight,

$$\bar{M} = \sum_\alpha Y_\alpha M_\alpha, \tag{5.97}$$

where Y_α are the gas phase mass fractions ans M_α are the molecular weights of the gaseous species.

The solid phase species equations solve for the solid phase mass fractions, $Y_{\alpha,s}$, of species α

$$\underbrace{\frac{\partial(\bar{\rho}Y_\alpha)}{\partial t}}_{MASS} = \underbrace{\left(\dot{\omega}'''_{f,\alpha} - \dot{\omega}'''_{d,\alpha}\right)}_{SRC}. \tag{5.98}$$

where $\dot{\omega}'''_{f,\alpha}$ is the formation rate of species α and $\dot{\omega}'''_{d,\alpha}$ is its destruction rate. The gas-phase species equations solve for the gas-phase mass fraction $Y_{\alpha,g}$ of species α

$$\underbrace{\frac{\partial(\psi\rho_g Y_{\alpha,g})}{\partial t}}_{MASS} + \underbrace{\frac{\partial(\rho_g u_{j,g} Y_{\alpha,g})}{\partial x_j}}_{ADV} + \underbrace{\frac{\partial q^{Y,g}_{\alpha j}}{\partial x_j}}_{DIFF} = \underbrace{\dot{\omega_{\alpha,g}}}_{SRC} \tag{5.99}$$

No source terms are included in this model. The gas-phase species diffusion flux vector $q^{Y,g}_{\alpha j}$ is modeled as

$$q^{Y,g}_{\alpha j} = -\psi\rho_g D_{\alpha,g}\frac{\partial Y_{\alpha,g}}{\partial x_j}, \tag{5.100}$$

where $D_{\alpha,g}$ is the gas-phase mass diffusivity for species α.

Enthalpy equations are used to solve for temperature, and OMD type problems allow for different temperature fields in the gas and solid phases, so these are described separately here.

The solid phase enthalpy equation for the solid phase temperature $\bar{T}$ is given by

$$\underbrace{\frac{\partial(\bar{\rho}c_p\bar{T})}{\partial t}}_{MASS} = \underbrace{-\frac{\partial q_j^h}{\partial x_j}}_{DIFF} + \underbrace{h_{cv}\left(T_g - \bar{T}\right)}_{SRC} \tag{5.101}$$

There is no advection term here since the solid phase is stationary. The solid phase energy diffusive flux vector $q_j^{h,g}$ is given by Fourier's law,

$$q_j^{h,g} = -\kappa\frac{\partial\bar{T}}{\partial x_j}, \tag{5.102}$$

where κ is the thermal conductivity.

The gas-phase enthalpy equation within a porous region, to be solved for the gas-phase enthalpy (and corresponding temperature) $h_g = c_pT_g$, is

$$\underbrace{\frac{\partial(\psi\rho_g h_g)}{\partial t}}_{MASS} + \underbrace{\frac{\partial(\rho_g u_{j,g} h_g)}{\partial x_j}}_{ADV} = \underbrace{-\frac{\partial q_j^{h,g}}{\partial x_j}}_{DIFF} + \underbrace{\left(\frac{\partial\psi p_g}{\partial t}\right)}_{SRC} + \underbrace{h_{cv}\left(\bar{T} - T_g\right)}_{SRC} \tag{5.103}$$

where h_g is the mixture-averaged gas-phase enthalpy, h_{cv} is the volumetric heat transfer coefficient which gets set outside of the solid/gas phases of the material definition, $\bar{T}$ is the porous solid-phase temperature and T_g is the gas phase temperature. The gas-phase energy diffusive flux vector $q_j^{h,g}$ is modeled as

$$q_j^{h,g} = -\psi\rho_g D_g\frac{\partial h_g}{\partial x_j}, \tag{5.104}$$

where D_g is the mixture-averaged gas-phase mass diffusivity. The compressible pressure source term for the enthalpy equation has been derived in the context of Darcy flow. The volumetric transfer source accounts for the heat transfer between the solid and gas phases.

5.11.3 Porosity Models

There are several models for porosity in the porous flow capabilities in Aria. The following list provides a brief description for each.

Mesh Deforming

In the mesh (or solid) deforming porosity model, the expression for the porosity is given as

$$\phi = 1.0 - \frac{b(1.0 - \phi_0)}{\det(\mathrm{F})} \tag{5.105}$$

where ϕ_0 is the initial porosity, b is the Biot coefficient, and F is the mesh deformation gradient.

Rock Compressible

For the Rock Compressible porosity model, the porosity is given as

$$\phi = \phi_0(1.0 + Cr(p - p_{ref})) \tag{5.106}$$

where again, ϕ_0 is the initial porosity, Cr is a rock compressibility coefficient, p is the pore pressure, and p_{ref} is a reference pore pressure defined in the input parameters.

Coussy

For the Coussy porosity model, the expression for porosity is

$$\phi = \phi_0 + \frac{1}{M}(p - p_{ref}) + b\epsilon_u \tag{5.107}$$

where M represents the formation compressibility, $\epsilon_u = \text{div}(\boldsymbol{u})$ is the divergence of the solid deformation vector, b is the Biot coefficient, p is the pore pressure, and p_{ref} is a reference pore pressure.

5.12 Brinkman Momentum

The Brinkman (Brinkman-Forchheimer) momentum equation is a porous media specialization of the fluid momentum equation 5.19 given by

$$\frac{\partial}{\partial t}\left(\frac{\rho}{\varphi}\boldsymbol{v}\right) + \frac{\rho}{\varphi^2}\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v} - \boldsymbol{\nabla}\cdot\boldsymbol{T} + \left[\frac{\rho\hat{c}}{\sqrt{k}}\|\boldsymbol{v}\| + \frac{\mu}{k}\right]\boldsymbol{v} - g = \boldsymbol{0} \tag{5.108}$$

where φ is the matrix porosity, $\boldsymbol{T}$ is the fluid stress tensor $\boldsymbol{g}$ is a body force and μ is termed the flowing liquid viscosity. The bracketed terms are often referred to as the Forchheimer or inertial flow contribution and the Darcy term respectively. In the limit of low velocities the equation reduces to a balance of pressure and a scaled velocity and Darcy's law for a single component fluid is recovered [12]. This equation is most useful in cases where a clear fluid domain is bounded by a porous media. In most applications of this equation the hydrostatic contribution will be negligible but is retained here for completeness. We construct the G/FEM residual form of 5.108 by contracting with the unit coordinate vector in the k-direction, $\boldsymbol{e}_k$, multiplying by the weight function ϕ^i and integrating over the volume. Using the vector identiy $(\boldsymbol{\nabla}\cdot\boldsymbol{T})\cdot\boldsymbol{e}_k\phi^i = \boldsymbol{\nabla}\cdot(\boldsymbol{T}\cdot\boldsymbol{e}_k\phi^i) - \boldsymbol{T}^t : \boldsymbol{\nabla}(\boldsymbol{e}_k\phi^i)$ and integrating by parts gives

$$R^i_{m,k} = \int_{\text{V}}\left[\frac{\partial}{\partial t}\left((\frac{\rho}{\varphi}\boldsymbol{v}) + \frac{\rho}{\varphi^2}\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v} + \left[\frac{\rho\hat{c}}{\sqrt{k}}\|\boldsymbol{v}\| + \frac{\mu}{k}\right] - g\right)\cdot\boldsymbol{e}_k\phi^i + \boldsymbol{T}^t : \boldsymbol{\nabla}\left(\boldsymbol{e}_k\phi^i\right)\right]\,\text{dV} - \int_{\text{S}}\boldsymbol{n}\cdot\boldsymbol{T}\cdot\boldsymbol{e}_k\phi^i\,\text{dS} = 0 \tag{5.109}$$

Here we note that the viscosity associated with the stress tensor $\boldsymbol{T}$ is not the viscosity of the fluid but instead a so-called Brinkman viscosity μ_B. In Aria, each term in 5.109 is specified separately as identified in equation 5.15.

$$R^i_{m,k} = \underbrace{\int_{\text{V}}\frac{\partial}{\partial t}(\frac{\rho}{\varphi}\boldsymbol{v})\cdot\boldsymbol{e}_k\phi^i\,\text{dV}}_{\text{MASS}} + \underbrace{\int_{\text{V}}\frac{\rho}{\varphi^2}\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}\cdot\boldsymbol{e}_k\phi^i\,\text{dV}}_{\text{ADV}} - \underbrace{\int_{\text{V}}\boldsymbol{g} - \left[\frac{\rho\hat{c}}{\sqrt{k}}\|\boldsymbol{v}\| + \frac{\mu}{k}\right]\boldsymbol{v}\cdot\boldsymbol{e}_k\phi^i\,\text{dV}}_{\text{SRC}}$$
$$+ \underbrace{\int_{\text{V}}\boldsymbol{T}^t : \boldsymbol{\nabla}\left(\boldsymbol{e}_k\phi^i\right)\,\text{dV}}_{\text{DIFF}} - \int_{\text{S}}\boldsymbol{n}\cdot\boldsymbol{T}\cdot\boldsymbol{e}_k\phi^i\,\text{dS} = 0 \tag{5.110}$$

5.13 Lubrication Equation

Reynolds' lubrication equation is a reduced-order model for fluid flow in thin, confined regions. This equation is commonly used for manufacturing applications, such as bearings, coating processes, and nanomanufacturing processes. Lubricating flows also occur in high-speed machinery, such as pumps. It is only defined for reduced-order elements, such as shells.

The lubrication equation implemented here is given by

$$-\frac{\partial h}{\partial t}+\boldsymbol{\nabla}\cdot\boldsymbol{q}+\boldsymbol{B}_u\cdot\boldsymbol{\nabla}h_u-\boldsymbol{B}_l\cdot\boldsymbol{\nabla}h_l=0 \tag{5.111}$$

where $\boldsymbol{q}$ is the lubrication flow rate, given by

$$\boldsymbol{q}=-\frac{h^3}{k\mu}\boldsymbol{\nabla}p_{\text{lub}}+\frac{h}{2}(\boldsymbol{B}_u+\boldsymbol{B}_l). \tag{5.112}$$

In these equations, the independent variable is the lubrication pressure, p_{lub}. Other variables are the lubrication region height (thickness) h, velocity of the upper confining surface $\boldsymbol{B}_u$, velocity of the lower confining surface $\boldsymbol{B}_l$, the upper wall height h_u, the lower wall height h_l, turbulent parameter k, and viscosity μ. The lubrication height is defined as $h=h_u-h_l+\boldsymbol{n}\cdot\boldsymbol{d}$, where $\boldsymbol{n}$ is the shell normal vector and $\boldsymbol{d}$ is the mesh displacement. This equation is detailed in [13].

Using equation 5.111, the G/FEM residual form is

$$R_V^i=\underbrace{-\int_{\text{V}}\frac{\partial h}{\partial t}\phi^i\,\text{dV}}_{\text{MASS}}+\underbrace{\int_{\text{V}}\boldsymbol{q}\cdot\boldsymbol{\nabla}\phi^i\,\text{dV}}_{\text{DIFF}}+\underbrace{\int_{\text{V}}(\boldsymbol{B}_u\cdot\boldsymbol{\nabla}h_u-\boldsymbol{B}_l\cdot\boldsymbol{\nabla}h_l)\phi^i\,\text{dV}}_{\text{BOUND}}+\int_{\text{S}}\boldsymbol{n}\cdot\boldsymbol{q}\phi^i\,\text{dS}=0 \tag{5.113}$$

5.14 Stress Tensor Projection Equation

A projection equation is defined as an equation where a derived quantity at the interior Gauss points is evaluated and projected to be a solution unknown at the nodal points. The stress tensor projection equation projects the momentum stresses, tau, without the pressure term, to the nodal points. Projecting the momentum stress smooths out the momentum stress tensor and allows for a dot product to be carried out on the projected field, which is needed for least squares stabilization schemes. The solution variable, τ_{ab}, is calculated from 5.114.

$$R_V^i=\underbrace{-\int_{\text{V}}(\tau_{ab}-src\_\tau_{ab})\phi^i\,\text{dV}=0}_{\text{DEF}} \tag{5.114}$$

τ_{ab} is a tensor variable. For 2D problems, ab stands for XX, XY, YX, and YY. For 3D problems ab stands for XX, XY, XZ, YX, YY, YZ, ZX, ZY, and ZZ. The source term in the equation $src_{\tau ab}$ refers to the momentum stress without the pressure diagonal term.

$$src\_\tau_{xx}=2\mu\frac{du}{dx}-\frac{2}{3}(\mu-\lambda)(\boldsymbol{\nabla}\cdot\boldsymbol{v})$$

$$src\_\tau_{yy}=2\mu\frac{dv}{dy}-\frac{2}{3}(\mu-\lambda)(\boldsymbol{\nabla}\cdot\boldsymbol{v})$$

$$src\_\tau_{zz}=2\mu\frac{dw}{dz}-\frac{2}{3}(\mu-\lambda)(\boldsymbol{\nabla}\cdot\boldsymbol{v})$$

$$src\_\tau_{xy} = src\_\tau_{yx} = \mu(\frac{du}{dy} + \frac{dv}{dx})$$

$$src\_\tau_{xz} = src\_\tau_{zx} = \mu(\frac{du}{dz} + \frac{dw}{dx})$$

$$src\_\tau_{yz} = src\_\tau_{zy} = \mu(\frac{dv}{dz} + \frac{dw}{dy})$$

5.15 Potential Projection Equation

This equation is used to create a potential field which minimizes difference between the potential, ψ, and the potential source, $\mathbf{S}$, in the least-squares sense

$$(\boldsymbol{\nabla}\psi - \mathbf{S})^2 = 0. \tag{5.115}$$

This equation was derived to create a hydrostatic potential field, so the most common usage is where $\mathbf{S} = \rho_s \mathbf{g}$.

Equation 5.115 is integrated by parts, resulting in the residual

$$R^i = \underbrace{\int_{\mathrm{V}} \boldsymbol{\nabla}\psi \cdot \boldsymbol{\nabla}(\boldsymbol{e}_k \phi^i)\,\mathrm{dV}}_{\mathrm{DEF}} - \underbrace{\int_{\mathrm{V}} \mathbf{S} \cdot \boldsymbol{\nabla}(\boldsymbol{e}_k \phi^i)\,\mathrm{dV}}_{\mathrm{SRC}} \tag{5.116}$$

5.16 Notes on Solid Mechanics

Some of the standard references on solid mechanics include [14], [15], [4] and [5]. As is often the case, the mathematical notion used through-out these texts is different in many cases and this is often a source of confusion. Here, we'll lay out some basic definitions in our notation and, when possible, give the notation used in these other texts.

In what follows, $\boldsymbol{x}$ is the position vector of a material particle in the *deformed* or *current* spatial configuration and $\boldsymbol{X}$ is the position vector of a material particle in the *undeformed* or *initial* or *reference* configuration. The displacement vector $\boldsymbol{d}$ is the difference between these to states[1] viz. $\boldsymbol{x} = \boldsymbol{X} + \boldsymbol{d}$.

We will make extensive use of the gradients of these fields and so it is important to distinguish between gradients in the reference configuration and the current configuration. Gradients in the current configuration are denoted $\boldsymbol{\nabla}$ in Gibbs notation or $\partial\ /\partial x_i$ in index notation. Gradients in the reference configuration are denoted $\boldsymbol{\nabla}_X$ in Gibbs notation or $\partial\ /\partial X_i$ in index notation[2].

Next, we define the deformation gradient $\boldsymbol{F}$

$$\begin{aligned} \boldsymbol{F} &\equiv \boldsymbol{\nabla}_X \boldsymbol{x}^t && (5.117)\\ &= \boldsymbol{\nabla}_X \boldsymbol{X}^t + \boldsymbol{\nabla}_X \boldsymbol{d}^t && (5.118)\\ &= \boldsymbol{I} + \boldsymbol{\nabla}_X \boldsymbol{d}^t && (5.119) \end{aligned}$$

[1] $\boldsymbol{d}$ is denoted $\boldsymbol{u}$ in [14], [15], [4], and [5].
[2] In [5] $\boldsymbol{\nabla}_X$ is denoted $\boldsymbol{\nabla}_0$.

where the superscript ^{t} denotes the transpose operator[3]. The inverse deformation gradient[4], $\boldsymbol{F}^{-1}$, is also useful and can be computed as

$$\begin{aligned} \boldsymbol{F}^{-1} &\equiv \boldsymbol{\nabla} \boldsymbol{X}^t && (5.120) \\ &= \boldsymbol{\nabla} \boldsymbol{x}^t - \boldsymbol{\nabla} \boldsymbol{d}^t && (5.121) \\ &= \boldsymbol{I} - \boldsymbol{\nabla} \boldsymbol{d}^t. && (5.122) \end{aligned}$$

The determinants of $\boldsymbol{F}$ and $\boldsymbol{F}^{-1}$ are denoted J and J^{-1} respectively and are often used in transformations between different stress definitions[5].

It's worth noting at this point that in Aria both gradient operators, $\boldsymbol{\nabla}$ and $\boldsymbol{\nabla}_X$, are available as Expression objects as are $\boldsymbol{F}$, $\boldsymbol{F}^{-1}$, J and J^{-1}.

The Green or Green-Lagrange strain tensor $\boldsymbol{E}$ is defined[6] as

$$\begin{aligned} \boldsymbol{E} &\equiv \frac{1}{2}\left(\boldsymbol{F}^t \cdot \boldsymbol{F} - \boldsymbol{I}\right) && (5.123) \\ &= \frac{1}{2}\left(\boldsymbol{\nabla}_X \boldsymbol{d} + \boldsymbol{\nabla}_X \boldsymbol{d}^t + \boldsymbol{\nabla}_X \boldsymbol{d} \cdot \boldsymbol{\nabla}_X \boldsymbol{d}^t\right). && (5.124) \end{aligned}$$

The Green strain is a strain measure in the reference configuration and is suitable for large deformations and large rotations. The analogous Eulerian (or Almansi's) strain tensor $\boldsymbol{E}^*$ is defined[7] as

$$\begin{aligned} \boldsymbol{E}^* &\equiv \frac{1}{2}\left(\boldsymbol{I} - \boldsymbol{F}^{-t} \cdot \boldsymbol{F}^{-1}\right) && (5.125) \\ &= \frac{1}{2}\left(\boldsymbol{\nabla} \boldsymbol{d} + \boldsymbol{\nabla} \boldsymbol{d}^t - \boldsymbol{\nabla} \boldsymbol{d} \cdot \boldsymbol{\nabla} \boldsymbol{d}^t\right). && (5.126) \end{aligned}$$

The Eulerian strain is also a suitable strain measure for large deformations and rotations but is defined in the current configuration.

The Cauchy stress, $\boldsymbol{\sigma}$, is a stress measure defined in the current configuration as

$$\boldsymbol{\sigma} = \lambda E_{kk} \boldsymbol{I} + 2\mu \boldsymbol{E} \tag{5.127}$$

where E_{kk} denotes the trace of $\boldsymbol{E}$ and λ and μ are the Lamé coefficients. This constitutive equation may also be augmented with some initial residual stress or a thermal stress,

$$\boldsymbol{\sigma} = \lambda E_{kk} \boldsymbol{I} + 2\mu \boldsymbol{E} - \beta \left(T - T_\circ\right) \boldsymbol{I} + \boldsymbol{\sigma}_r. \tag{5.128}$$

Here β is related to the coefficient of thermal expansion, T is the temperature, $T_\circ$ is the reference temperature of the solid and $\boldsymbol{\sigma}_r$ is the residual stress.

For large deformations and large rotations Aria uses the second Piola-Kirchhoff stress which is defined in the reference configuration and is related to the Cauchy stress as

$$\boldsymbol{S} = J \boldsymbol{F}^{-1} \cdot \boldsymbol{\sigma} \cdot \boldsymbol{F}^{-t}. \tag{5.129}$$

[3] In [15] this is called the *conjugate dyadic* and is denoted with a subscript c. In [14] the quantity $\boldsymbol{\nabla}_X \boldsymbol{x}^t$ is denoted $\boldsymbol{x} \overleftarrow{\boldsymbol{\nabla}}_X$ where the arrow over the gradient operator denotes the direction of the operation.

[4] In [15] $\boldsymbol{F}^{-1}$ is denoted $\boldsymbol{H}$.

[5] Sometimes J is expressed as a ratio of the densities between the reference and current configurations, $\rho_\circ/\rho$.

[6] In [15] $\boldsymbol{E}$ is denoted $\boldsymbol{L}_G$ and is called the *Lagrangian strain*.

[7] In [15] $\boldsymbol{E}^*$ is denoted $\boldsymbol{E}_A$.

The reverse transformation is readily given by

$$\boldsymbol{\sigma} = J^{-1}\boldsymbol{F}\cdot\boldsymbol{S}\cdot\boldsymbol{F}^{t}. \tag{5.130}$$

Mathematically, the Cauchy stress $\boldsymbol{\sigma}$ is most conveniently expressed in terms of the Lamé coefficients λ, μ and β. In practice, however, it is more common to measure and report a different but related set of parameters: the Young's modulus E, the Poisson's ratio ν and the coefficient of thermal expansion α. (Note, the shear modulus $G = \mu$.) The relationship between these two sets of parameters is

$$2\mu = \frac{E}{(1+\nu)} \tag{5.131}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} = 2\mu\frac{\nu}{(1-2\nu)} \tag{5.132}$$

$$\beta = \frac{\alpha E}{(1-2\nu)} = \alpha(3\lambda+2\mu) \tag{5.133}$$

In Aria, there are separate `Expression_Name`s for the Cauchy stress and the second Piola-Kirchhoff stress expressions. In the input files, users provide their choice of constitutive relations in the material model specification, e.g.,

```
Begin Aria Material The_Material
   Density     = Constant rho = 2.33e-15
   Mesh Lambda = Constant lambda = 52810.30445
   Mesh Two Mu = Constant two_mu = 134426.2295
   Mesh Stress = Nonlinear_Elastic Reference_Frame=Moving
   Mesh Stress = Residual Sx=-11 Sy=-11
   Mesh Stress = Isothermal T=800 T_ref=450
End
```

In this Example, there will be three contributions to the stress in the mesh stress: nonlinear elasticity, a planar residual stress and an isotropic linear thermal stress. Internally, Aria contains a separate expression for transforming these Cauchy stresses into Piola-Kirchhoff stresses, viz.

$$\boldsymbol{S} = J\boldsymbol{F}^{-1}\cdot\left(\sum_i \boldsymbol{\sigma}_i\right)\cdot\boldsymbol{F}^{-t}. \tag{5.134}$$

Note that second Piola-Kirchhoff stresses are not specified in the input file – only Cauchy stresses are. Aria automatically creates an Expression to compute the transform in (5.134).

5.17 Units and Unit Conversions

Aria make no a priori specification concerning the units of each term. However, as with all engineering codes, errors associated with unit conversions are quite easy to do. In some situations, with just one or two driving forces playing a role in a calculation, nondimensionalization of the equations can lead to a simplification of the problem statement (and to increased solution robustness due to proper scaling of the terms in the equations). However, in complicated cases with multiple competing forces and rate constants, sticking to unit systems to specify all quantities frequently leads to less errors in engineering calculations, and also leads to the ability to incorporate third party library packages for specification of source terms and transport properties which necessarily presume to employ units systems in their application programming interfaces (API). The next section discusses the SI units system, and its application for reacting flow and electromagnetic applications.

5.17.1 SI Units

Quantity	Name	Abbreviation
length	meter (metre)	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K
amount of substance	kmole	kmol
luminous intensity	candela	cd

Table 5.1. Fundamental SI units.

discuss electromagnetic unit specification, e.g., Gauss's law.

Factor	Prefix	Abbreviation
10^{24}	yotta	Y
10^{21}	zetta	Z
10^{18}	exa	E
10^{15}	peta	P
10^{12}	tera	T
10^{9}	giga	G
10^{6}	mega	M
10^{3}	kilo	k
10^{2}	hecto	h
10^{1}	deca	da
10^{-1}	deci	d
10^{-2}	centi	c
10^{-3}	milli	m
10^{-6}	micro	μ
10^{-9}	nano	n
10^{-12}	pico	p
10^{-15}	femto	f
10^{-18}	atto	a
10^{-21}	zepto	z
10^{-24}	yocto	y

Table 5.2. SI magnitude prefixes.

Quantity	Unit	Definition
Frequency	hertz	$Hz = 1/s$
Force	newton	$N = m \cdot kg/s^2$
Pressure, stress	pascal	$Pa = N/m^2 = kg/m \cdot s^2$
Energy, work, quantity of heat	joule	$J = N \cdot m = m^2 \cdot kg/s^2$
Power, radiant flux	watt	$W = J/s = m^2 \cdot kg/s^3$
Quantity of electricity, electric charge	coulomb	$C = s \cdot A$
Electric potential	volt	$V = W/A = m^2 \cdot kg/s^3 \cdot A$
Capacitance	farad	$F = C/V = s^4 \cdot A^2/m^2 \cdot kg$
Electric resistance	ohm	$Omega = V/A = m^2 \cdot kg/s^3 \cdot A^2$
Conductance	siemens	$S = A/V = s^3 \cdot A^2/m^2 \cdot kg$
Magnetic flux	weber	$Wb = V \cdot s = m^2\ kg/s^2 \cdot A$
Magnetic flux density, magnetic induction	tesla	$T = Wb/m^2 = kg/s^2 \cdot A$
Inductance	henry	$H = Wb/A = m^2\ kg/s^2 \cdot A^2$
Luminous flux	lumen	$lm = cd \cdot sr$
Illuminance	lux	$lx = lm/m^2 = cd \cdot sr/m^2$
Activity (ionizing radiations)	becquerel	$Bq = 1/s$
Absorbed dose	gray	$Gy = J/kg = m^2/s^2$
Dynamic viscosity	pascal second	$Pa \cdot s = kg/m \cdot s$
Moment of force	meter newton	$N \cdot m = m^2 \cdot kg/s^2$
Surface tension	newton per meter	$N/m = kg/s^2$
Heat flux density, irradiance	watt per square meter	$W/m^2 = kg/s^3$
Heat capacity, entropy	joule per kelvin	$J/K = m^2 \cdot kg/s^2 \cdot K$
Specific heat capacity, specific entropy	joule per kilogram kelvin	$J/kg\ K = m^2/s^2 \cdot K$
Specific energy	joule per kilogram	$J/kg = m^2/s^2$
Thermal conductivity	watt per meter kelvin	$W/m \cdot K = m \cdot kg/s^3 \cdot K$
Energy density	joule per cubic meter	$J/m^3 = kg/m\ s^2$
Electric field strength	volt per meter	$V/m = m \cdot kg/s^3 \cdot A$
Electric charge density	coulomb per cubic meter	$C/m^3 = s \cdot A/m^3$
Electric displacement, electric flux density	coulomb per square meter	$C/m^2 = s \cdot A/m^2$
Permittivity	farad per meter	$F/m = s^4 \cdot A^2/m^3 \cdot kg$
Permeability	henry per meter	$H/m = m \cdot kg/s^2 \cdot A^2$
Molar energy	joule per kmol	$J/kmol = m^2 kg/s^2 \cdot kmol$
Molar entropy, molar heat capacity	joule per kmol kelvin	$J/kmol\ K = m^2 kg/s^2 \cdot K \cdot kmol$
Exposure (ionizing radiations)	coulomb per kilogram	$C/kg = s \cdot A/kg$
Absorbed dose rate	gray per second	$Gy/s = m^2/s^3$

Table 5.3. SI derived units and their definitions.

Quantity	Unit
erg	$1\ erg = 10^{-7}\ J$
dyne	$1\ dyn = 10^{-5}\ N$
poise	$1\ P = 1\ dyn \cdot s/cm^2 = 0.1\ Pa \cdot s$
stokes	$1\ St = 1\ cm^2/s = 10^{-4}\ m^2/s$
gauss	$1\ G = 10^{-4}\ T$
oersted	$1\ Oe = (1000/(4\ pi))\ A/m$
maxwell	$1\ Mx = 10^{-8}\ Wb$
stilb	$1\ sb = 1\ cd/cm^2 = 10^4\ cd/m^2$
phot	$1\ ph = 10^4\ lx$

Table 5.4. CGS derived units and their definitions.

5.17.2 Common Units and Conversion Factors

1 g·/s^2 = 1 dyn = 10^{-5} kg·m/s^2 = 10^{-5} N
1 g·/s^2 = 7.2330×10^{-5} lb$_m$·ft/s^2 (poundal)
1 lb$_f$ = 4.4482 N
1 g·/s^2 = 2.2481×10^{-6}lb$_f$

Table 5.5. Units and conversion factors for force.

1 bar = 10^5 Pa = 10^5 N/m^2
1 psia = 1 lb$_f$/in^2
1 psia = 2.0360 in Hg at 0 °C
1 psia = 2.311 ft H_2O at 70 °F
1 psia = 51.715 mm Hg at 0 °C (ρ_{Hg} = 13.5955 g/cm^3)
1 atm = 14.696 psia = 1.01325×10^5N/m^2 = 1.01325 bar
1 atm = 760 mm Hg at 0 °C = 1.01325×10^5 Pa
1 atm = 29.921 in Hg at 0 °C
1 atm = 33.90 ft H_2O at 4 °C

Table 5.6. Units and conversion factors for pressure and stress.

1 cp = 10^{-2} g/cm· s = 10^{-2} Poise
1 cp = 2.4191 lb$_m$/ft·h
1 cp = 6.7197×10^{-4} lb$_m$/ft·s
1 cp = 10^{-3} Pa·s = 10^{-3} kg/m·s = 10^{-3} N·/m^2
1 cp = 2.0886×10^{-5} lb$_f$ · s/ft^2
1 Pa·s = 1 N·s/m^2 = 1 kg/m·s = 1000 cp = 0.67197 lb$_m$/ft·s

Table 5.7. Units and conversion factors for viscosity.

1 g/cm^3 = 1000 kg/m^3 = 62.43 lb$_m$/ft^3
1 g/cm^3 = 8.345 lb$_m$/U.S. gal
1 lb$_m$/ft^3 = 16.0185 kg/m^3

Table 5.8. Units and conversion factors for density.

1 btu/h·ft·°F = 4.1365×10^{-3} cal/s·cm·°C
1 btu/h·ft·°F = 1.73073 W/m·K

Table 5.9. Units and conversion factors for thermal conductivity.

1 btu/h·ft^2·°F = 1.3571×10^{-4} cal/s·cm^2·°C
1 btu/h·ft^2·°F = 5.6783×10^{-4} W/cm^2·°C
1 btu/h·ft^2·°F = 5.6783 W/m^2·°C
1 kcal/h·m^2·°F = 0.2048 btu/h·ft^2·°F

Table 5.10. Units and conversion factors for heat-transfer coefficients.

Chapter 6

Equation Specification

This chapter documents the EQ line commands within the current version of Aria. EQ line commands add equations and independent variables to Aria's specification of the equation set to be solved for within each region. The equation is also associated with a field variable here that becomes part of the solution vector for Aria. EQ commands occur in Aria's input file within Region blocks.

The EQ command line adds an equation to be solved for on a particular *MESH_PART* . The format is as follows

EQ equation FOR *DOF* on *MESH_PART* using *INTERP* with $\mathrm{TERM}_0 \ldots \mathrm{TERM}_n$

Equation is the string identifier for the individual equations listed in the previous chapter. The format for the equation string identifiers is listed in a subsection of Chapter 2. The *DOF* keyword specifies the independent unknown that is solved for in order to satisfy the equation. Normally, it is a strict function of the equation keyword. In other words, the temperature is the only valid *DOF* entry if the energy equation is being solved. *MESH_PART* is usually the name of an active element block in the finite element model. Unfortunately, if an equation is to be solved on the entire finite element model, this means that there must be multiple EQ keywords for each element block defined in the mesh.

INTERP defines the finite element interpolation to be used. Currently the valid entries for this keyword are P0, P1, Q1, Q2 and Q2S with standard volumes, e.g. hexahedron, tetrahedron, quadrilateral and triangle. These keywords imply the interpolation and polynomial order for the solution field where P denotes (polynomial) element interpolation and Q denotes the nodal interpolation (Q1 - linear, Q2 - quadratic and Q2S for near-quadratic , serendipity). We note that no syntactical distinction is made between element topology (i.e. triangle and quadrilateral or tetrahedron and hexahedron), that is, the syntax indicates only the level of interpolation. The association of element topology with the equation is handled internal to the code and allowing the specification of interpolation independent of the topology enables the use of superparametric elements (i.e. quadratic geometry with linear interpolation).

Parametric coordinates of shell and bar elements differ from those of conventional bulk volume elements (e.g. hexahedron and tetrahedron) since their meshed geometric description lacks a thickness or area dimension. This missing dimension (SHELL THICKNESS and BAR AREA) can be provided for each material independent of the meshed discretization. Hence when defining equations on shell elements the *INTERP* keywords previously mentioned are paired (e.g. Q1P0) according to interpolation in the shell plane and interpolation through the shell thickness. For example Q1P0 represents linear nodal interpolation in the plane of the shell and constant interpolation through the shell thickness. Similarly for bar elements an *INTERP* specification of Q1P0 represents linear nodal interpolation along the length of the bar and constant interpolation in the bar section. In sets of coupled equations, arbitrary combinations of interpolations are sometimes not permitted for solution variables appearing in coupling terms of an equation as interpolation must be consistent in any given equation.

$\mathtt{TERM}_n$ refer to the broad categories for the terms in a general advection-diffusion continuity equation. Each term in the equation must be explicitly turned on for it to appear in the conservation equation. Admissible values of TERM are MASS, LUMPED_MASS, ADV, DIFF, SRC, and XFER. While MASS, LUMPED_MASS, ADV, DIFF, SRC terms imply the activation of mathematical operators appearing in an

equation, XFER implies that a solution field values for this equation are being provided externally so the equation can still be thought of as belonging to the overall equation set.

Provisions are made for supplying multiple equation contributions for DIFF. These contributions are usually activated via settings in the Aria Materials command block 4.1. Multiple contributions can also be defined for SRC by adding additional source command lines 11. In order to accommodate different definitions of velocity for ADV the user can request the velocity to be provided from another equation, from transfer or from other available velocity models 6.27.

All equations are assumed to be formulated in Cartesian coordinates. For two-dimensional problems, axisymmetric formulations are available as well as the Cartesian forms. Invocation of the axisymmetric option is described in the Model Definition chapter of this manual 3.1 in the FINITE ELEMENT MODEL section.

6.1 EQ CONTINUITY

`EQ CONTINUITY[_<Subindex>][_<Phase>] for` ***`DOF`*** `ON` ***`MESH_PART`*** `USING` ***`INTERP`*** `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the continuity equation 5.4.

Summary The only admissible value for ***`DOF`*** is PRESSURE.

Admissible values of ***`INTERP`*** are Q_1, Q_2, Q_{2S}, P_1, P_0, for standard volume elements and Q_1P_0, Q_2P_0, $Q_{2S}P_0$, Q_1Q_1 and Q_1Q_2 for shell elements.

Admissible values of $\texttt{TERM}_n$ are MASS, LUMPED_MASS, DIV, ADV, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The ***`MESH_PART`*** must be an active element block.

Parent Block(s) `ARIA_REGION`

6.2 EQ CURRENT

`EQ CURRENT[_<Subindex>][_<Phase>] for` ***`DOF`*** `ON` ***`MESH_PART`*** `USING` ***`INTERP`*** `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the electrical current equation 5.33.

Summary The only admissible value for ***`DOF`*** is VOLTAGE.

Admissible values of ***`INTERP`*** are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The ***`MESH_PART`*** must be an active element block.

Parent Block(s) `ARIA_REGION`

6.3 EQ CHARGE_DENSITY

EQ CHARGE_DENSITY[_<Subindex>][_<Phase>] for *DOF* ON *MESH_PART* USING *INTERP* with $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Beta Capability: The charge density capability is not well tested and should be used with caution.

Description Activates the charge density equation 5.36.

Summary The only admissible value for *DOF* is CHARGE_DENSITY.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are MASS, LUMPED_MASS, ADV, DIFF, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.4 EQ ENERGY

EQ ENERGY[_<Subindex>][_<Phase>] for *DOF* ON *MESH_PART* USING *INTERP* with $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the energy conservation transport equation 5.9.

Summary Admissible values for *DOF* are TEMPERATURE and ENTHALPY.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}, for standard volume elements and Q_1P_0, Q_2P_0, Q_1Q_1 and Q_1Q_2 for shell elements.

Admissible values of $\texttt{TERM}_n$ are MASS, LUMPED_MASS, ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section[5.3] of the manual.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.5 EQ LEVEL_SET

EQ LEVEL_SET[_<Subindex>] for *DOF* ON *MESH_PART* USING *INTERP* with $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the level set (distance function) equation. This equation is often accompanied by additional commands described in 29.

Summary The only admissible value for *DOF* is LEVEL_SET.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of TERM$_n$ are MASS, LUMPED_MASS, ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.6 EQ EXTENSION_SPEED

EQ EXTENSION_SPEED[_<Subindex>] for *DOF* ON *MESH_PART* USING *INTERP* with TERM$_0$...TERM$_n$

Description Activates an equation for extension speed, which is used by the level set equation (29) to define the interface speed.

Summary The only admissible value for *DOF* is EXTENSION_SPEED.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of TERM$_n$ are DEF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.7 EQ MASS_BALANCE

EQ MASS_BALANCE[_<Subindex>] for *DOF* ON *MESH_PART* USING *INTERP* with TERM$_0$...TERM$_n$

Description The mass balance equation for porous media. See general porous flow information in section 5.11

Summary Admissible value for *DOF* are PRESSURE, DENSITY, and MASS_FRACTION.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of TERM$_n$ are MASS, LUMPED_MASS, ADV, UPWIND_ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.8 EQ MESH

EQ MESH[_<Subindex>] for *DOF* ON *MESH_PART* USING *INTERP* with $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the pseudo-solid mesh equation 5.22.

Beta Capability: Material models for this capability are under active development hence this equation should be used with caution.

Summary The only admissible value for *DOF* is MESH_DISPLACEMENTS.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are DIFF, REF_DIFF, TSTRAIN, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

If both the MESH and SOLID equations are defined in the same region, then the TALE mesh motion algorithm is activated automatically. In this case, the MESH equation must be defined on all blocks in the region. More details of the TALE method can be found in section 5.6.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.9 EQ MOMENTUM

EQ MOMENTUM[_<Subindex>][_<Phase>] for *DOF* ON *MESH_PART* USING *INTERP* with $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Beta Capability: The momentum equation is tested for low Reynolds numbers, $R_e < 10$ but should be used with caution beyond that range.

Description Activates the fluid momentum equation 5.19.

Summary The only admissible value for *DOF* is VELOCITY.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are MASS, LUMPED_MASS, ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.10 EQ POROUS_SPECIES

EQ POROUS_SPECIES[_<Subindex>] for *DOF* ON *MESH_PART* USING *INTERP* with $\texttt{TERM}_0$...$\texttt{TERM}_n$

Description The species transport equation porous media. See general porous flow information in section 5.11

Summary Admissible value for *DOF* are PRESSURE, SPECIES, and SPECIES_FRACTION.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are MASS, LUMPED_MASS, ADV, UPWIND_ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Additionally, one can choose whether to integrate the advection term by parts by setting the flag INTEGRATE ADVECTION BY PARTS = false in the POROUS_FLOW_OPTIONS block of SOLUTION_OPTIONS. The default is true.

Parent Block(s) ARIA_REGION

6.11 EQ POROUS_ENTHALPY

EQ POROUS_ENTHALPY[_<Subindex>] for *DOF* ON *MESH_PART* USING *INTERP* with $\texttt{TERM}_0$...$\texttt{TERM}_n$

Description The enthalpy transport equation porous media. See general porous flow information in section 5.11

Summary The only admissible value for *DOF* is TEMPERATURE.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are MASS, LUMPED_MASS, ADV, UPWIND_ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Additionally, one can choose whether to integrate the advection term by parts by setting the flag INTEGRATE ADVECTION BY PARTS = false in the POROUS_FLOW_OPTIONS block of SOLUTION_OPTIONS. The default is true.

Parent Block(s) ARIA_REGION

6.12 EQ POTENTIAL

`EQ POTENTIAL[_<Subindex>][_<Phase>] for` *`DOF`* `ON` *`MESH_PART`* `USING` *`INTERP`* `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the potential projection equation for the potential, which is defined in equation 5.115.

Summary The only admissible value for *`DOF`* is POTENTIAL.

Admissible values of *`INTERP`* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are DEF and SRC, and both terms must be active for this equation to behave properly. The value used in SRC must be a separately defined source equation.

The *`MESH_PART`* must be an active element block.

Parent Block(s) `ARIA_REGION`

6.13 EQ SHEAR

`EQ SHEAR[_<Subindex>][_<Phase>] for` *`DOF`* `ON` *`MESH_PART`* `USING` *`INTERP`* `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the shear-rate definition/intermediate equation.

Summary The only admissible value for *`DOF`* is GAMMA_DOT.

Admissible values of *`INTERP`* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are DEF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *`MESH_PART`* must be an active element block.

Parent Block(s) `ARIA_REGION`

6.14 EQ SOLID

`EQ SOLID[_<Subindex>][_<Phase>] for` *`DOF`* `ON` *`MESH_PART`* `USING` *`INTERP`* `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Beta Capability: Material models for this capability are under active development hence this equation should be used with caution.

Description Activates the solid momentum equation 5.22.

Summary The only admissible value for *DOF* is SOLID_DISPLACEMENTS.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of TERM$_n$ are DIFF, TSTRAIN, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

If both the MESH and SOLID equations are defined in the same region, then the TALE mesh motion algorithm is activated automatically. In this case, the reference configuration for the SOLID equation is converted to the TALE reference frame. More details of the TALE method can be found in section 5.6.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.15 EQ SPECIES

EQ SPECIES[_<Subindex>][_<Phase>] for *DOF* ON *MESH_PART* USING *INTERP* with TERM$_0$...TERM$_n$

Beta Capability: The species equation is tested for low Reynolds numbers, $R_e < 10$ but should be used with caution beyond that range.

Description Activates the species transport equation 5.13.

Summary The only admissible value for *DOF* is SPECIES.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

Admissible values of TERM$_n$ are MASS, LUMPED_MASS, ADV, DIFF, SRC, FRACBAL and XFER. With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual. The FRACBAL term may not be included with other terms.

Some other things to note about species equations in Aria.

- The FRACBAL term may be assigned to any species number.
- Species numbers in Aria are arbitrary; they may start at any value and need not be continuous.
- For the species fraction balances, Aria will automatically detect all species that are present in the problem and include them in the balance.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.16 EQ SP

`EQ SP` *`ORDER`* `for` *`DOF`* `ON` *`MESH_PART`* `USING` *`INTERP`* `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Beta Capability: The spherical harmonics capability is not well tested and should be used with caution.

Description Specifies the Spherical Harmonics equation set to solve for Participating Media Radiation (PMR). User provides the order, degree of freedom associated with the equation, the element type and the mesh part (element block) on which to apply the equation 23.2.

Summary Admissible values for *ORDER* are odd values (1, 3, 5, ...).

The only admissible value for *`DOF`* is `angular_intensity`.

The *`MESH_PART`* must be an active element block.

Admissible values of *`INTERP`* are Q_1, Q_2, Q_{2S}

Admissible value of $\texttt{TERM}_n$ is SRC.

Parent Block(s) `ARIA_REGION`

6.17 EQ SUSPENSION

`EQ SUSPENSION[_<Subindex>][_<Phase>] for` *`DOF`* `ON` *`MESH_PART`* `USING` *`INTERP`* `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Beta Capability: The suspension modeling capability is not well tested and should be used with caution.

Description Activates the suspension transport equation 5.39.

Summary The only admissible value for *`DOF`* is PHI.

Admissible values of *`INTERP`* are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *`MESH_PART`* must be an active element block.

Parent Block(s) `ARIA_REGION`

6.18 EQ VOLTAGE

`EQ VOLTAGE[_<Subindex>][_<Phase>] for` ***DOF*** `ON` ***MESH_PART*** `USING` ***INTERP*** `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Description Activates the voltage equation (electric-displacement formulation) 5.30. See also the CURRENT equation.

Summary The only admissible value for ***DOF*** is VOLTAGE.

Admissible values of ***INTERP*** are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The ***MESH_PART*** must be an active element block.

Parent Block(s) `ARIA_REGION`

6.19 EQ BRINKMAN_MOMENTUM

`EQ BRINKMAN_MOMENTUM[_<Subindex>][_<Phase>] for` ***DOF*** `ON` ***MESH_PART*** `USING` ***INTERP*** `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Beta Capability: The porous media capability is under active development hence the Brinkman momentum equation should be used with caution.

Description Activates the Brinkman momentum equation 5.108.

Summary The only admissible value for ***DOF*** is VELOCITY.

Admissible values of ***INTERP*** are Q_1, Q_2, Q_{2S}.

Admissible values of $\texttt{TERM}_n$ are MASS, LUMPED_MASS, ADV, DIFF, SRC, and XFER.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The ***MESH_PART*** must be an active element block.

Parent Block(s) `ARIA_REGION`

6.20 EQ LUBRICATION

`EQ LUBRICATION[_<Subindex>][_<Phase>] for` ***DOF*** `ON` ***MESH_PART*** `USING` ***INTERP*** `with` $\texttt{TERM}_0 \ldots \texttt{TERM}_n$

Beta Capability: The lubrication modeling capability is under active development hence this equation should be used with caution.

Description Activates the Reynolds' lubrication equation 5.111.

Summary The only admissible value for *DOF* is LUBRICATION_PRESSURE.

Admissible values of *INTERP* is Q_1P_0.

Admissible values of TERM$_n$ are MASS, DIFF, and BOUND.

For problems with fluid-structural interactions, the lubrication equation needs to be defined on all blocks, including 3-D blocks adjoining the shell lubrication region. In order to accommodate this, there is an additional term, DEF, which applies the equation $p = 0$ on adjoining 3-D blocks where this is used. The other terms should not be used on 3-D blocks. It is weighted so that it does not affect the residual on the shell blocks.

The *MESH_PART* must be an active shell element block.

Parent Block(s) ARIA_REGION

6.21 EQ STRESS_TENSOR_PROJECTION

EQ Stress_Tensor_Projection[_<Subindex>][_<Phase>] for *DOF* ON *MESH_PART* USING *INTERP* with TERM$_0$...TERM$_n$

Description Activates the Stress Tensor Projection equation.

Summary The only admissible value for *DOF* is Stress_Tensor.

Admissible values of *INTERP* are Q_1, Q_2, Q_{2S}.

The only admissible values of TERM$_n$ are DEF.

With the exception of XFER these terms are described in the Equations Aria Solves section 5 of the manual.

The *MESH_PART* must be an active element block.

Parent Block(s) ARIA_REGION

6.22 ELASTICITY FORMULATION

ELASTICITY FORMULATION = PLANE *TYPE*

Description Assigns the elasticity formulation *TYPE* for two-dimensional problems involving the MESH and SOLID equation 5.22.

Summary Allowable formulation PLANE *TYPE* is PLANE STRESS or PLANE STRAIN.

Parent Block(s) `ARIA_REGION`

6.23 PRESSURE STABILIZATION

`PRESSURE STABILIZATION IS` *TYPE* `WITH SCALING =` C

Description Prescribe a stabilization technique for solving the `MOMENTUM` 5.19 and `CONTINUITY` 5.4 equations with equal order interpolation.

Summary Aria supports both PSPG (Pressure Stabilized Petrov-Galerkin) and PSPP (Pressure Stabilized Pressure Projection) stabilization techniques for solving the `MOMENTUM` and `CONTINUITY` equations with equal order interpolation.

Valid options for the *TYPE* specification are `NO_STABILIZATION`, `PSPG_CONSTANT`, `PSPG_LOCAL`, `PSPG_GLOBAL` and `PSPP_CONSTANT`.

`NO_STABILIZATION` disables any stabilization.

`PSPP_CONSTANT` results in the recently developed stabilization technique of [16] and [17].

In the PSPG forms of stabilization, introduced by [18], terms from the momentum equation are added to the continuity equation scaled by a multiplier, α. The exact form of the multiplier depend on a global Reynolds number that is defined as

$$Re \equiv \frac{\rho|\boldsymbol{v}|\langle h\rangle}{2\mu} \tag{6.1}$$

Here, ρ is the density, μ is the viscosity, $|\boldsymbol{v}|$ is a velocity scale and $\langle h\rangle$ is an element length scale. Armed with Re, the stabilization multiplier α is defined in one of two ways.

$$Re \leq 3 \quad : \quad \alpha \equiv \frac{\tau\langle h\rangle^2}{12\mu} \tag{6.2}$$

$$Re > 3 \quad : \quad \alpha \equiv \frac{\tau\langle h\rangle}{2\rho|\boldsymbol{v}|} \tag{6.3}$$

NB: Currently, Aria always uses the low-Reynolds number for of α, as defined in (6.2). The `PSPG_LOCAL` method computes $|\boldsymbol{v}|$ and $\langle h\rangle$ within each element. The `PSPG_GLOBAL` method computes $|\boldsymbol{v}|$ and $\langle h\rangle$ as averages over all of the elements with the `MOMENTUM` equation defined. The `PSPG_CONSTANT` gives $|\boldsymbol{v}|$ and $\langle h\rangle$ a value of 1 (one) and just uses the scale factor.

Parent Block(s) `ARIA_REGION`

6.24 SAVE RESIDUALS

`SAVE RESIDUALS =` *MODE*

Description Causes Aria to save the residuals to a field with the prefix `residual->`, e.g., `residual->Temperature`. This will be done for all fields (though we could make it a per-field option). Valid choices are `OFF` (default), `BEFORE_BCS` and `AFTER_BCS`.

Summary Causes Aria to save the residuals to a field with the prefix `residual->`, e.g., `residual->Temperature`. This will be done for all fields (though we could make it a per-field option). Valid choices are `OFF` (default), `BEFORE_BCS` and `AFTER_BCS`.

For the choice `BEFORE_BCS` the residuals will be saved at the point in the assembly process where the primary equations have been assembled but prior to the assembly of any boundary conditions or distinguishing conditions. For the choice of `AFTER_BCS` the residuals will be saved after all BCs and distinguishing conditions have been applied. The default, `OFF`, is to not save the residuals.

This feature is only applicable when using the `NEWTON` nonlinear solution strategy.

Parent Block(s) `ARIA_REGION`

6.25 INTEGRATION RULE

`INTEGRATION RULE for` *`block_name`* `=` *`INT`*

Description Overrides the default integration rule for the equations defined on *`block_name`*.

Summary Overrides the default integration rule for the equations defined on *`block_name`*.

Parent Block(s) `ARIA_REGION`

6.26 MESH GROUP

```
MESH GROUP alias_name = mesh_entity1 .  .  mesh_entityN
MESH GROUP alias_name = group1 + mesh_entity
MESH GROUP alias_name = group1 - group2
MESH GROUP alias_name = block_1 to block_N
```

Description Assigns aliased grouping of selected mesh entities (block, surface or nodelist). These aliases can then be used in defining equations, source terms, initial conditions, Dirichlet boundary conditions, surface flux conditions, postprocessing operations and the `INTEGRATION RULE` command.

In order to simplify the specification of mesh groups for large models, sum, difference and range operators are provided.

Summary Only mesh entities can be aliased within a group and a mesh entity can appear in several groups. Mesh entities aliased within the `FINITE ELEMENT MODEL` command block can appear within `MESH GROUP` and are treated in the same way as the mesh entity itself. Additionally mesh entities with names supplied from the input file can also be referenced

By default the aliases *`all_blocks`* and *`all_volumes`* reference all active element blocks in the mesh and the *`all_surfaces`* alias references all active surfaces in the mesh. Default aliases are also defined for element blocks by topology and number of nodes, e.g. Hexahedron_8_blocks.

The first mesh_entity can be an alias to an existing `MESH GROUP` provided subsequent entries define addition or subtraction to the group using (+, plus, and, -, minus, less) to define the

operation used to compose an alias. For example:

MESH GROUP *alias_name* = *all_blocks* - *block_8*

An additional form of the command defines a range of element blocks. In this case the group range is defined using (to, thru or through) to specify the first and last element blocks in the range based upon block number. Note that Mesh Group commands are parsed sequentially, thus command line ordering must be honored when using one Mesh Group to define another Mesh Group.

Parent Block(s) ARIA_REGION

6.27 ADVECTION VELOCITY

ADVECTION VELOCITY for *block_name* = *STRING*

Description Defines the velocity model to be used for the equations defined on *block_name*.

Summary Velocity model can be XFER or any other vector velocity model defined in the Region.

Parent Block(s) ARIA_REGION

Chapter 7

Data for Material, BC, IC, SRC

In many numerical simulations the need often arises to define quantities that involve the same numerical values and are accessible in user defined functions (i.e. user plugins and user subroutines). To facilitate this usage Aria provides a means by which one can collectively define problem dependent real and integer data as named members of a "Data Block", where named data members can be defined either as independent values or as arrays. The utility of Data Block information in setting up problem descriptions is that rather than referencing individual data values, the data collection can be referenced as a whole using the name of the parent data block. This provides a convenient means of reusing data when defining material behavior, boundary conditions and initial conditions with user defined functions for a particular class of problem.

For material parameters the scope of Data Block information is at Domain scope (i.e. outside of the Procedure) hence the corresponding Data Block is specified at the domain scope. For boundary conditions, source terms and initial conditions the scope of Data Block information lies within an Aria Region. Each Data Block must have a unique name and will be referenced by that name within the problem setup. Reference to individual Data Block members will occur only within user defined functions.

Because Data Block values are named then within user defined functions code individual members of the Data Block can be easily referenced by name. This feature is particularly useful in both Calore style user subroutines and in user plugin utilities. The usage of Data Blocks with Calore-style user subroutines is detailed in the Chapter on Thermal Analysis 33.19. The Data Block can be used in two distinct ways, at the code feature level or at the Region level. An association between the Data Block and a code feature or the Region is established with a **Use Data Block DataBlockName** command.

Note that an analogous capability **aprepro** exists for substituting fixed or computed data values directly in the input file. Using aprepro one can perform these substitutions anywhere within the input file.

7.1 Data Block

Scope: Aria Region

```
Begin Data DataBlockName
    Real {= | parameter_name | =} value...
    Integer {= | parameter_name | =} value...
End
```

Summary Parameters for Data Blocks within Aria. The DataBlockName must be referenced within the definition of an initial condition, volumetric source, boundary condition or material property Calore_User_Sub in order to be available for use within a user subroutine.

The DataBlockName may also be used to define parameters for all equation line commands by specifying USE DATA BLOCK DataBlockName as the first parameter argument on the equation. When used in this manner, the typed name-value pairs of the Data Block are assigned as the parameters for the equation.

7.1.1 Real

Scope: Data Block

`Real {= | parameter_name | =}` *`value...`*

Parameter	Value	Default
values	`real...`	

Summary Specifies the Real data values for this data block.

7.1.2 Integer

Scope: Data Block

`Integer {= | parameter_name | =}` *`value...`*

Parameter	Value	Default
values	`integer...`	

Summary Specifies the Integer data values for this data block.

Chapter 8

Initial Conditions

This chapter documents the initial condition, IC, line commands within the current version of Aria. The IC simply represents an initialization the Field variable. Technically speaking initial conditions have meaning only in time-dependent partial differential equations. However, within the Aria code the IC command line results in an initialization of Field values used in the assembly of a Jacobian matrix. Thus the IC is equally applicable for assigning starting values in a steady solution. Note that in the case of steady solutions the Aria output will contain two "time" steps, an initialization output plane, and the steady solution output. A distinction between the meaning of these two steps is particularly important when using a steady solution result to provide the initial condition from file for a transient problem.

Within the input file initial conditions command lines will appear at the Region scope as illustrated below. Note that input specification of initial conditions on portions of the FEM mesh can be simplified by aggregation of element blocks into Mesh Groups 6.26.

```
.
.
.
Begin Procedure My_Aria_Procedure
   .
   .
   Begin Aria Region My_Region
    .
    .
    IC related commands
    .
    .
   End

End
.
.
```

Beta Capability: These initial conductions are defined in a paradigm that will eventually be deprecated.

8.1 IC CIRC_X

IC CIRC_X AT *MESH_PART DOF* AMP = Ω

Description Initial boundary condition for the x component of a vector variable with constant tangential magnitude along circles of radius $r(x, y)$ defined on the mesh entity.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* according to the relation

$$DOF = -\Omega r(x, y) \sin\theta. \tag{8.1}$$

The rotation Ω is assumed to be about the vector $(0, 0, 1)$ passing through point $(0, 0, 0)$.

Parent Block(s) ARIA_REGION

8.2 IC CIRC_Y

IC CIRC_Y AT *MESH_PART DOF* AMP = Ω

Description Initial boundary condition for the y component of a vector variable with constant tangential magnitude along circles of radius $r(x, y)$ defined on the mesh entity.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* according to the relation

$$DOF = \Omega r(x, y) \cos\theta. \tag{8.2}$$

The rotation Ω is assumed to be about the vector $(0, 0, 1)$ passing through point $(0, 0, 0)$.

Parent Block(s) ARIA_REGION

8.3 IC CONSTANT

IC CONST AT *MESH_PART DOF* = *REAL*

Description Constant initial condition.

Summary Sets *DOF* to the provided constant value on mesh entity associated with the name *MESH_PART*.

Parent Block(s) ARIA_REGION

8.4 IC COUETTE_X

IC COUETTE_X AT *MESH_PART DOF* PARAMS = r_i r_o Ω

Description Initial condition for x component of a vector variable in a Couette device with inner radius r_i, outer radius r_o and driven at angular velocity Ω, that varies spatially over a mesh entity.

Summary Sets *DOF* to vary spatially over the nodeset associated with the name *MESH_PART* according to the relation

$$DOF = -\Omega\, r(x,y)\frac{r_i r_o}{r_o^2 - r_i^2}\left(\frac{r_o^2}{r} - r\right)\sin\theta \tag{8.3}$$

The rotation Ω is assumed to be about the vector $(0,0,1)$ passing through point $(0,0,0)$.

Parent Block(s) ARIA_REGION

8.5 IC COUETTE_Y

IC COUETTE_Y AT *MESH_PART DOF* PARAMS = r_i r_o Ω

Description Initial condition for x component of a vector variable in a Couette device with inner radius r_i, outer radius r_o and driven at angular velocity Ω, that varies spatially over a mesh entity.

Summary Sets *DOF* to vary spatially over the nodeset associated with the name *MESH_PART* according to the relation

$$DOF = \Omega\, r(x,y)\frac{r_i r_o}{r_o^2 - r_i^2}\left(\frac{r_o^2}{r} - r\right)\cos\theta \tag{8.4}$$

The rotation Ω is assumed to be about the vector $(0,0,1)$ passing through point $(0,0,0)$.

Parent Block(s) ARIA_REGION

8.6 IC COUETTE_SH

IC COUETTE_SH AT *MESH_PART DOF* PARAMS = r_i r_o Ω

Description Initial condition for generalized shear rate in a Couette device with inner radius r_i, outer radius r_o and driven at angular velocity Ω, that varies spatially over a mesh entity.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* according to the relation

$$DOF = \frac{\Omega}{r^2(x,y)}\frac{r_i r_o}{r_o^2 - r_i^2}\left(\frac{1}{r(x,y)} - \frac{r(x,y)}{r_o^2}\right) \tag{8.5}$$

The rotation Ω is assumed to be about the vector $(0,0,1)$ passing through point $(0,0,0)$.

Parent Block(s) ARIA_REGION

8.7 IC READ_FILE

`IC READ_FILE` *DOF* `=` *STRING* `AT TIME` *REAL*

Description This IC command will initialize the field *DOF* with values from the field with the name given by the *STRING* argument in the input mesh database.

Summary This IC command will initialize the field *DOF* in the current analysis with values from the input mesh database containing variable *STRING*. For example, if the mesh database as specified in the `FINITE ELEMENT MODEL` command block (3.1) contains a vector field named `U` then one could use the file to initialize a velocity field by setting *DOF* to `VELOCITY` and the *STRING* argument to `U`.

In cases where the field *DOF* has a several components (i.e. *DOF* is a vector field) and the input mesh database contains the vector components defined as scalar fields, the command must be called once for each component. Here the *DOF* is specified as *DOF*_component.

A typical usage example would be initialize a transient calculation from a previously performed steady-state calculation in the following way. The steady-state calculation would be performed and its output variables saved to some output file. In most cases one might employ a copy of the output file to be used as a mesh file for the transient calculation. Finally, one would use the file IC command line to tie the variable named "FileVariableName" in the mesh file to the variable named *DOF* in the current simulation.

When the mesh database file contains multiple timesteps and the TIME option is not supplied, the default behavior will be to extract the requested field from the last timeplane in the file. The TIME option must be used if the initial condition is to be extracted from any specific database timeplane other than the last timeplane. In using the TIME option the database time step nearest to the specified time value will be used for the assignment. For an initial condition taken from a steady-state results file, the optional TIME parameter should be omitted.

Steady solution results are sometimes written to file in two steps. When this command is used to initialize values from a steady solution one must select the solution file step according to the content of the steady solution file results. If the file contains two solution steps then one should employ the time option and select the time corresponding to the second step.

CAVEAT: Current implementations do not allow restricting this initialization by mesh subsets. e. g. If one wishes to initialize the temperature, then all nodes in the mesh will be initialized.

NOTE: Those wishing to perform initializations as part of a formal restart are referred to `RESTART DATA` services (25.1).

Parent Block(s) `ARIA_REGION`

8.8 IC LINEAR

`IC LINEAR AT` *MESH_PART* *DOF* `COEF =` C_0 C_1 C_2 C_3

Description Initial condition that varies spatially over a mesh entity in a linear fashion.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* in the following manner

$$DOF = C_0 + C_1 x + C_2 y + C_3 z. \tag{8.6}$$

Parent Block(s) ARIA_REGION

8.9 IC PARAB

IC PARAB AT *MESH_PART* *DOF* COEF = C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9

Description Initial condition that varies spatially over a mesh entity in a parabolic fashion.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* in the following manner

$$DOF = C_0 + C_1 x + C_2 y + C_3 z + C_4 xy + C_5 xz + C_6 yz + C_7 x^2 + C_8 y^2 + C_9 z^2. \tag{8.7}$$

Parent Block(s) ARIA_REGION

8.10 ANNEAL MESH ON STARTUP

ANNEAL MESH ON STARTUP

Description With the command in the Region block Aria will search for a displacements field and, if found, will apply the displacements to the model and physical coordinates.

Summary With the command in the Region block Aria will search for a displacements field and, if found, will apply the displacements to the model and physical coordinates.

Parent Block(s) ARIA_REGION

Chapter 9

Boundary Conditions

Boundary conditions are needed in order to define a well-posed boundary value problem as described by the FEM equations. These boundary conditions are often denoted as Dirichlet, Neumann or Robin conditions. Dirichlet or essential boundary conditions for a solution field stem directly from the governing equations for a particular physics. In Aria, Neumann and Robin conditions are generally categorized as flux boundary conditions.

This section documents primarily the native Aria boundary condition, BC, line commands within the current version of the code. Recall from a previous chapter outlining the general FEM that the method gives rise to boundary terms 5.3, i.e. various flux boundary conditions. In what follows, Dirichlet boundary conditions are first described 9.1 followed by flux boundary condition descriptions 9.2.

Boundary conditions are defined at the Region scope of the input file as illustrated below.

```
Begin Sierra myJob
  .
  .
  Begin Procedure My_Aria_Procedure
     .
     .
     Begin Aria Region My_Region
      .
      .
      BC related commands
      .
      .
     End

  End
  .
End Sierra myJob
```

9.1 BC Dirichlet

Dirichlet or essential boundary conditions for a solution field stem directly from the governing equations for a particular physics. Here the boundary conditions are strongly enforced by substituting the user prescribed evaluation of solution unknown into the system residual and the corresponding solution increment equal to zero. In Aria default enforcement of the Dirichlet boundary condition involves a matrix row modification. Optional methods of Dirichlet BC enforcement can be specified through the linear solver interface
tt BC Enforcement command line.

The boundary condition specification corresponds to the solution variable or degree-of-freedom (DOF) for a

given equation as it appears in the EQ command line. As an example in the EQ command line:

```
EQ Energy for Temperature on block_2 using Q1 with Diff mass
```

`Temperature` is the DOF that used in specification of the Dirichlet boundary condition whereas `Energy` is the equation to which the condition is applied.

For thermal analysis another interface for Dirichlet BC specification is described in the Thermal Analysis chapter 33.1. The advantage of this approach is that the syntax is fully supported within the Sandia Analysis Workbench (SAW).

Dirichlet boundary conditions are applied at a surface of the physical problem domain. Within the context of Sierra Mechanics that surface corresponds to a surface (e.g. surface_3) or a nodelist (e.g. nodelist_2). In what follows the `Constant`, `Encore Function`, `Global`, `User Function`, `Polynomial`, `User Field` and `Exponential` conditions must be applied to a surface. The remainder of the Dirichlet BCs listed can be applied to either a surface or a nodelist.

Note that stylistically the syntax for `Constant`, `Encore Function`, `Global`, `User Function`, `Polynomial`, `User Field` and `Exponential` conditions differ from the remaining Dirichlet BC command lines. As an example:

```
BC Dirichlet BC for Energy on surface_3 = constant value = 273.0
```

implicitly refers to a `Temperature` boundary condition. This is in contrast to the remaining Dirichlet BC command lines:

```
BC Const Dirichlet at surface_3 temperature = 273.0
```

where the DOF is explicitly stated.

9.1.1 Generic: Constant

Scope: Equation System

`Generic: Constant`
`For` *`Dof Name`* `[ {of|species|subindex}` *`Species`* `|{in|material_phase}` *`MaterialPhase`* `]{@|at|for|in|on|over}` *`Mesh Extent Name`* `= Constant [ Using Data Specification` *`Data Spec Name`* `][ Value =` *`value`* `]`

Parameter	Value	Default
Dof Name	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
value	real	undefined

Summary Constant value

9.1.2 Generic: Encore Function

Scope: Equation System

`Generic: Encore Function`
`For` *`Dof Name`* `[ {of|species|subindex}` *`Species`* `|{in|material_phase}` *`MaterialPhase`* `]{@|at|for|in|on|over}` *`Mesh Extent Name`* `= Encore_Function [ Using Data Specification` *`Data Spec Name`* `][ Name =` *`name`* `|Result_Name =` *`result_name`* `|Eval_Type =` *`eval_type`* `]`

Parameter	Value	Default
Dof Name	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
result_name	"string"	undefined
eval_type	"string"	undefined

Summary Value from an Encore function

9.1.3 Generic: Global

Scope: Equation System

```
Generic: Global
For Dof Name [ {of|species|subindex} Species |{in|material_phase} MaterialPhase ]{@|at|
for|in|on|over} Mesh Extent Name = Global [ Using Data Specification Data Spec Name ][ Global_Name
= global_name ]
```

Parameter	Value	Default
Dof Name	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
global_name	"string"	undefined

Summary Value from a global variable

9.1.4 Generic: User Function

Scope: Equation System

```
Generic: User Function
For Dof Name [ {of|species|subindex} Species |{in|material_phase} MaterialPhase ]{@|at|
for|in|on|over} Mesh Extent Name = User_Function [ Using Data Specification Data Spec Name
][ Name = name |X = x |X_Multiplier = x_multiplier |Multiplier = multiplier |Toggle = toggle
]
```

Parameter	Value	Default
Dof Name	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
x	"string"	undefined
x_multiplier	real	undefined
multiplier	real	undefined
toggle	"string"	undefined

Summary Value from a user function

9.1.5 Generic: Polynomial

Scope: Equation System

```
Generic: Polynomial
For Dof Name [ {of|species|subindex} Species |{in|material_phase} MaterialPhase ]{@|at|
for|in|on|over} Mesh Extent Name = Polynomial [ Using Data Specification Data Spec Name ][
Variable = variable |Order = order |Variable_Offset = variable_offset |C0 = c0 |C1 = c1
|C2 = c2 |C3 = c3 |C4 = c4 |C5 = c5 |C6 = c6 |C7 = c7 |C8 = c8 ]
```

Parameter	Value	Default
Dof Name	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
variable	"string"	undefined
order	integer	undefined
variable_offset	real	undefined
c0	"string"	undefined
c1	real	undefined
c2	real	undefined
c3	real	undefined
c4	real	undefined
c5	real	undefined
c6	real	undefined
c7	real	undefined
c8	real	undefined

Summary Value from a polynomial function

9.1.6 Generic: User Field

Scope: Equation System

```
Generic: User Field
For Dof Name [ {of|species|subindex} Species |{in|material_phase} MaterialPhase ]{@|at|
for|in|on|over} Mesh Extent Name = User_Field [ Using Data Specification Data Spec Name ][
Name = name |Scaling = scaling |Global_Var = global_var ]
```

Parameter	Value	Default
Dof Name	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
scaling	real	undefined
global_var	"string"	undefined

Summary Value from a user field

9.1.7 Generic: Exponential

Scope: Equation System

```
Generic:  Exponential
For Dof Name [ {of|species|subindex} Species |{in|material_phase} MaterialPhase ]{@|at|
for|in|on|over} Mesh Extent Name = Exponential [ Using Data Specification Data Spec Name
][ Variable = variable |Constant = constant |Multiplier = multiplier |Exponent = exponent
]
```

Parameter	Value	Default
Dof Name	string	undefined
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
Data Spec Name	string	undefined
variable	"string"	undefined
constant	real	undefined
multiplier	real	undefined
exponent	real	undefined

Summary Value from an exponential function

9.1.8 BC Dirichlet = BC CIRC_X

```
BC CIRC_X DIRICHLET AT MESH_PART DOF AMP = Ω
```

Description Dirichlet boundary condition for the x component of a vector variable with constant tangential magnitude along circles of radius $r(x, y)$ defined on the mesh entity.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* according to the relation

$$DOF = -\Omega r(x, y) \sin\theta. \tag{9.1}$$

The rotation Ω is assumed to be about the vector $(0, 0, 1)$ passing through point $(0, 0, 0)$.

Parent Block(s) ARIA_REGION

9.1.9 BC Dirichlet = BC CIRC_Y

```
BC CIRC_Y DIRICHLET AT MESH_PART DOF AMP = Ω
```

Description Dirichlet boundary condition for the y component of a vector variable with constant tangential magnitude along circles of radius $r(x, y)$ defined on the mesh entity.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* according to the relation

$$DOF = \Omega r(x, y) \cos\theta. \tag{9.2}$$

The rotation Ω is assumed to be about the vector $(0, 0, 1)$ passing through point $(0, 0, 0)$.

Parent Block(s) ARIA_REGION

9.1.10 BC Dirichlet = BC CONST

BC CONST DIRICHLET AT *MESH_PART DOF* = *REAL*

Description Constant Dirichlet condition.

Summary Sets *DOF* to the provided constant value on mesh entity associated with the name *MESH_PART*.

Parent Block(s) ARIA_REGION

9.1.11 BC Dirichlet = BC DISTRIBUTION FACTOR

BC DISTRIBUTION FACTOR DIRICHLET AT *MESH_PART DOF* SCALING = *REAL*

Description Dirichlet condition from node distribution factors.

Summary Sets *DOF* to the a scaled value of the distribution factors present on mesh entity associated with nodeset of name *MESH_PART*.

Parent Block(s) ARIA_REGION

9.1.12 BC Dirichlet = BC LINEAR

BC LINEAR DIRICHLET AT *MESH_PART DOF* COEF = C_0 C_1 C_2 C_3

Description Dirichlet boundary condition that varies spatially over a mesh entity in a linear fashion.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART*in the following manner

$$DOF = C_0 + C_1 x + C_2 y + C_3 z. \tag{9.3}$$

Parent Block(s) ARIA_REGION

9.1.13 BC Dirichlet = BC LINEAR_IN_TIME

BC LINEAR_IN_TIME DIRICHLET AT *MESH_PART DOF* COEF = C_0 C_1

Description Dirichlet boundary condition whose value is a linear function in time.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* according to the relation

$$DOF = C_0 + C_1 t \tag{9.4}$$

Parent Block(s) ARIA_REGION

9.1.14 BC Dirichlet = BC PARAB

BC PARAB DIRICHLET AT *MESH_PART* *DOF* COEF = C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9

Description Dirichlet boundary condition for *DOF* that varies spatially over a mesh entity in a parabolic fashion.

Summary Sets *DOF* to the provided value on nodeset associated with the name *MESH_PART* in the following manner

$$DOF = C_0 + C_1 x + C_2 y + C_3 z + C_4 xy + C_5 xz + C_6 yz + C_7 x^2 + C_8 y^2 + C_9 z^2. \tag{9.5}$$

Parent Block(s) ARIA_REGION

9.1.15 BC Dirichlet = BC PERIODIC_LINEAR_IN_TIME

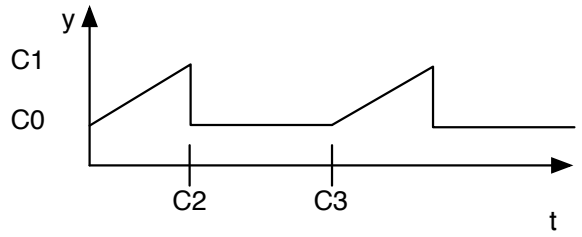
BC PERIODIC_LINEAR_IN_TIME DIRICHLET AT *MESH_PART* *DOF* COEF = C_0 C_1 C_2 C_3

Description Dirichlet boundary condition that provides a periodic linear (ramp) function in time over a portion of the local time period $\tau = C_3$.

Summary The value of *DOF* is given by a periodic linear (ramp) function in time, within a local time interval τ as illustrated in figure below.

$$DOF = \begin{cases} C_0 + C_1 t_p & t_p \le t_d \\ C_0 & otherwise \end{cases} \tag{9.6}$$

where $t_p = t - \tau \mathrm{Int}(t/\tau)$ is a local time period and $t_d = C_2$ is the dwell time within this time period. Note that $t_d < \tau$ in order for the boundary condition to be uniquely defined.



Parent Block(s) ARIA_REGION

9.1.16 BC Dirichlet = BC PERIODIC_STEP_IN_TIME

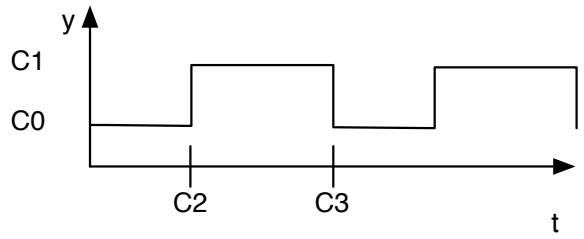
`BC PERIODIC_STEP_IN_TIME DIRICHLET AT` *`MESH_PART DOF`* `COEF =` C_0 C_1 C_2 C_3

Description Dirichlet boundary condition that provides a periodic step function in time over a portion of the local time period $\tau = C_3$.

Summary The value of *DOF* is given by a periodic step function in time, as illustrated in figure below.

$$DOF = \begin{cases} C_0 & t_p \leq t_d \\ C_1 & otherwise \end{cases} \tag{9.7}$$

where $t_p = t - \tau \text{Int}(t/\tau)$ is a local period time and $t_d = C_2$ is the dwell time within this time period. Note that $t_d < \tau$ in order for the boundary condition to be uniquely defined.



Parent Block(s) `ARIA_REGION`

9.1.17 BC Dirichlet = BC RAMP_LINEAR_IN_TIME

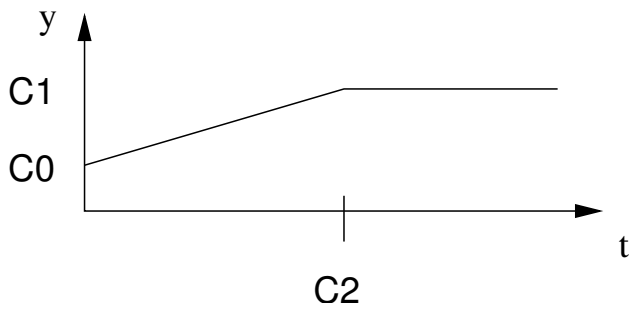
`BC RAMP_LINEAR_IN_TIME DIRICHLET AT` *`MESH_PART DOF`* `COEF =` C_0 C_1 C_2

Description Dirichlet boundary condition that provides a linear function in time over a range $t = 0$ to $t = C_2$ and constant value thereafter.

Summary The value of *DOF* is given by a linear function of time, as illustrated in figure below.

$$DOF = C_0 + (C_1 - C_0)\ \text{MIN}(t/C2, 1.0) \tag{9.8}$$

where C_0 is the initial value, C_1 is the final value and C_2 is the time at which the final value C_1 is achieved.



Parent Block(s) `ARIA_REGION`

9.1.18 BC Dirichlet = BC ROTATING_X

`BC ROTATING_X DIRICHLET AT` *`MESH_PART DOF`* `Omega =` Ω C_0 C_1 C_2

Description Dirichlet boundary condition that applies the x-component of a rotation in time about the z-axis.

Summary Sets *`DOF`* to the provided value on nodeset associated with the name *`MESH_PART`* according to the relation

$$\textit{DOF}\_X = X_0 \left(\cos \omega t - 1\right) - Y_0 \sin \omega t \tag{9.9}$$

The rotation Ω is assumed to be about the vector $(0, 0, 1)$ passing through point $(0, 0, 0)$.

Parent Block(s) `ARIA_REGION`

9.1.19 BC Dirichlet = BC ROTATING_Y

`BC ROTATING_X DIRICHLET AT` *`MESH_PART DOF`* `Omega =` Ω C_0 C_1 C_2

Description Dirichlet boundary condition that applies the y-component of a rotation in time about the z-axis.

Summary Sets *`DOF`* to the provided value on nodeset associated with the name *`MESH_PART`* according to the relation

$$\textit{DOF}\_Y = X_0 \sin \omega t + Y_0 \left(\cos \omega t - 1\right) \tag{9.10}$$

The rotation Ω is assumed to be about the vector $(0, 0, 1)$ passing through point $(0, 0, 0)$.

Parent Block(s) `ARIA_REGION`

9.1.20 BC Dirichlet = BC TRANSLATE

`BC TRANSLATE DIRICHLET AT` *`MESH_PART DOF`* `SCALE =` C_0

Description Dirichlet boundary condition for translating a value in time.

Summary Sets *`DOF`* to the provided value on nodeset associated with the name *`MESH_PART`* according to the relation

$$\textit{DOF} = C_0 t \tag{9.11}$$

Parent Block(s) `ARIA_REGION`

9.1.21 BC Dirichlet = BC UNIDIRECTIONAL_FLOW_X

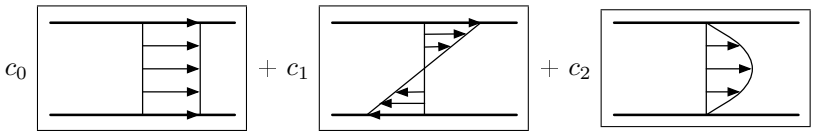
`BC UNIDIRECTIONAL_FLOW_X DIRICHLET AT` *`MESH_PART DOF`* `coef =` c_0 c_1 c_2

Description A specialized boundary condition for conveniently specifying inflow boundary conditions on simple geometries.

Summary This boundary condition is really meant for specifying inflow velocity profiles in a convenient way. The three coefficients provide the magnitudes of plug, shear and parabolic flow as a function of the y-coordinate. Specifically,

$$\textit{DOF} = c_0 + c_1 \frac{y - y_\circ}{H} + c_2 \left[1 - \left(\frac{y - y_\circ}{H} \right)^2 \right] \tag{9.12}$$

Here $y_\circ$ is the y-coordinate of the middle of the surface (sideset) or nodelist (nodeset) and H is the half-width of the surface or nodelist. Both $y_\circ$ and H are automatically calculated by Aria. This combination of flows is represented graphically as



Parent Block(s) `ARIA_REGION`

9.1.22 BC Dirichlet = BC UNIDIRECTIONAL_FLOW_Y

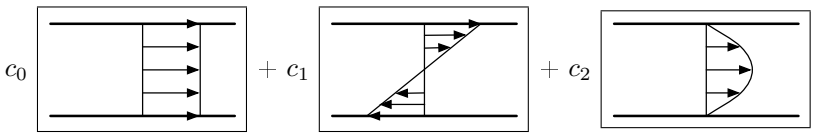
`BC UNIDIRECTIONAL_FLOW_Y DIRICHLET AT` *`MESH_PART DOF`* `coef =` c_0 c_1 c_2

Description A specialized boundary condition for conveniently specifying inflow boundary conditions on simple geometries.

Summary This boundary condition is really meant for specifying inflow velocity profiles in a convenient way. The three coefficients provide the magnitudes of plug, shear and parabolic flow as a function of the x-coordinate. Specifically,

$$\textit{DOF} = c_0 + c_1 \frac{x - x_\circ}{H} + c_2 \left[1 - \left(\frac{x - x_\circ}{H} \right)^2 \right] \tag{9.13}$$

Here $x_\circ$ is the x-coordinate of the middle of the surface (sideset) or nodelist (nodeset) and H is the half-width of the surface or nodelist. Both $x_\circ$ and H are automatically calculated by Aria. This combination of flows is represented graphically as



Parent Block(s) `ARIA_REGION`

9.1.23 BC Dirichlet = BC XFER

`BC XFER DIRICHLET AT` *`MESH_PART DOF`*

Description Constant Dirichlet condition where the value is set via a transfer.

Summary Allows *`DOF`* to set to a value that comes from a transfer operation on mesh entity associated with the name *`MESH_PART`*. Within the transfer operation the target (destination) Field should be defined as solution->*`DOF`* state NEW.

Example Assuming that a scalar Field for temperature is being provided from Transfer then use of the XFER BC could be defined as follows:

```
BEGIN PROCEDURE myProcedure
   .
  BEGIN ARIA REGION myRegion
    .
    BC Xfer Dirichlet at surface_2 Temperature
    .
  END ARIA REGION myRegion

  BEGIN transfer my_transfer
    .
    send field afield state none to solution->temperature state new
    .
  END transfer my_transfer
   .
END PROCEDURE myProcedure
```

Parent Block(s) `ARIA_REGION`

9.1.24 BC Dirichlet = BC USER FIELD

`BC USER FIELD DIRICHLET AT` *`MESH_PART DOF`* `= UserFieldName`

Description Dirichlet boundary condition to provide values as a function of time via a user defined Field.

Summary This Dirichlet boundary condition provides *`DOF`* values as a function of time from a user defined Field. Typically these values would be supplied by an Encore utility or from an external source via the Transfer utility.

Example Assuming that a scalar Field tf_BC is being provided from Transfer then use of the BC could be defined as follows:

```
BEGIN PROCEDURE myProcedure
   .
  BEGIN ARIA REGION myRegion
    .
    USER FIELD REAL NODAL SCALAR tf_BC on surface_2
    .
    BC User Field Dirichlet at surface_2 Temperature = tf_BC
    .
  END ARIA REGION myRegion

  BEGIN transfer my_transfer
    .
    send field afield state none to tf_BC state none
    .
  END transfer my_transfer
   .
END PROCEDURE myProcedure
```

Parent Block(s) ARIA_REGION

9.1.25 BC Dirichlet = BC USER_FUNCTION_IN_TIME

BC USER_FUNCTION_IN_TIME DIRICHLET AT *MESH_PART DOF* = UserFunctionName

---

Description Dirichlet boundary condition that provides values as a function of time via a user input function.

Summary Dirichlet boundary condition that provides values as a function of time via a user input function.

Example

```
Begin Sierra myJob
   .
  Begin Definition for Function T_BC_func
    Type is Piecewise Linear
    Column Titles Time Temperature
    Begin Values
      0 0
      10 50
    End
  End
  BEGIN PROCEDURE myProcedure
    .
    BEGIN ARIA REGION myRegion
     .
     BC User_Function_In_Time Dirichlet at surface_3 Temperature = T_BC_func
     .
    END ARIA REGION myRegion
   .
  END PROCEDURE myProcedure
End Sierra myJob
```

Parent Block(s) `ARIA_REGION`

9.2 BC FLUX

In Aria, Neumann and Robin conditions are categorized as flux boundary conditions applied to a surface of the problem domain. Here the flux boundary conditions are applied by performing a surface integration of the user prescribed evaluation to generate the appropriate system residual contributions.

The flux boundary condition specification corresponds to a given equation as it appears in the EQ command line. As an example in the EQ command line:

```
EQ Energy for Temperature on block_2 using Q1 with Diff mass
```

`Energy` is the equation to which the residual contributions will be applied.

For thermal analysis another form of Flux BC specification (Calore-style syntax) is described in the Thermal Analysis chapter 33.1. The advantage of this approach is that the syntax is fully supported within the Sandia Analysis Workbench (SAW).

Native Aria boundary conditions differ from those in the Calore-style in that they are succinctly contained within a single line rather than in a command block. In what follows the `Constant`, `Distribution_Factor`, `Encore_Function` and `User_Field` Flux conditions can be applied to any equation. The remainder of the Flux conditions listed apply to specific equations.

One additional difference between the native Aria and Calore-style flux boundary condition syntax is that the native Aria constant flux condition is defined based upon a mathematical convention rather than an engineering convention. By contrast, a constant flux condition in the Calore-style flux condition respects the engineering sign convention, i.e. energy input is positive while energy output is negative. Thus the native Aria constant flux sign convention is opposite that of the Calore-style convention. For other flux boundary conditions these differences are resolved internal code so that problems are always solved in a consistent manner. Nevertheless, users must be aware of the difference in these sign conventions when setting up thermal problems.

For both the native Aria and Calore-style syntax, specification of boundary conditions on portions of the FEM mesh can be simplified by aggregation of surface parts into Mesh Groups 6.26. Flux boundary conditions available in Aria are listed following a definition of the general syntax and the associated operation.

BC FLUX FOR *EQNAME* ON *MESH_PART* = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]
or
BC LUMPED_FLUX FOR *EQNAME* ON *MESH_PART* = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Neumann boundary condition that sets the surface normal flux of the degree of freedom associated with equation *EQNAME* to that provided by the specified *MODEL*.

Summary *MODEL* the supplies a diffusive flux $f_n \equiv \boldsymbol{n} \cdot \boldsymbol{f}$ in accordance with equation 5.3. I.e., it adds the surface integral

$$\int_{\textit{MESH\_PART}} \boldsymbol{n} \cdot \boldsymbol{f} \phi^i \, \mathrm{dS} \tag{9.14}$$

to the residual for equation *EQNAME*. See, e.g., q_n in equation 5.10.

When LUMPED_FLUX is used in place of FLUX, the integral will be evaluated at node locations, instead of the gauss points. This can be beneficial when $\boldsymbol{f}$ varies strongly along the surface dS, by preventing oscillations in the solution at the expense of integration accuracy for higher-order element types.

Parent Block(s) `ARIA_REGION`

9.2.1 BC FLUX = CONSTANT

Parameters
```
FLUX = REAL
```

Example
```
BC Flux for Energy on surface_10 = Constant Flux=3.14159
```

Description `FLUX` is the value of the constant flux where positive values indicate a loss, i.e., positive flux leaves the volume.

9.2.2 BC FLUX = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example

```
BC FLUX for Energy on surface_10 = Calore_User_Sub name=w80afftuser
type=element NR=2 R0=1000 R1=0.7
```

Description

`NAME` is the name of the user subroutine was registered with. `NR` and `NI` are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the `R`n (`I`n) parameter. **Note:** the parameters use zero based counting. The `TYPE` parameter denotes the API the user subroutine uses; currently Aria only supports types `ELEMENT` and `NODE`.

The optional `MULTIPLIER` parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

The following is an example user subroutine that provides a free-space radiation energy flux. Note that the flux sign convention is that positive values indicate a heat loss, i.e., positive flux leaves the volume.

Example

```
#include "math.h"
#include "Aria_Calore_User_Sub_Support.h"

Int
w80afftuser(UserQuery &                          user_query,
            Int                                  element_id,
            Int                                  num_elements,
            Int                                  num_points,
            Int                                  spatial_dimension,
            CoordinateElementIntegrationPoints   coordinate_array,
            TemperatureElementIntegrationPoints temperature_array,
            FluxElementIntegrationPoints         flux_array)
{

  RealArray1d rdat(2);

  user_query.getUserRealInstanceData(2, rdat.ptr());

  const Real Tref       = rdat(0);
  const Real emissivity = rdat(1);

  const Real sb_emissivity = 5.67E-8 * emissivity;
  const Real Tref4         = Tref * Tref * Tref * Tref;

  for(Int element=0; element < num_elements; ++element)
    {
      for(Int point=0; point < num_points; ++point)
        {
          const Real & T            = temperature_array(element,point);
          const Real T2             = T*T;
          flux_array(element,point) = sb_emissivity * (T2*T2 - Tref4);
        }
    }
  return 0;
}

extern "C"
void
#if defined(DYNAMIC_USER_PLUGIN)
dl_register()
#else
SIERRA_FORTRAN(fmwk_reg_user_subs)()
#endif
{
  sierra::Plugin::
    UserSubroutine<aucal_elem_sub_c>::registerFunction("w80afftuser",
                                                       &w80afftuser);
}
```

Description

To use this subroutine, you must add a line command at the domain level of your Aria input file that gives the name of the user subroutine source file, e.g.,

```
User Subroutine file = w80afftuser.C
```

Then, to build the executable with the user subroutine, you can use a command line like this,

```
sierra --make aria -i w80afftuser_aria.i
```

And, finally, the problem can be run like this,

```
sierra ./UserSubsProject/bin/aria -i w80afftuser_aria.i
```

Note that the `UserSubsProject` is created as a part of the build process.

9.2.3 BC FLUX = DISTRIBUTION_FACTOR

Parameters `MULTIPLIER = REAL`

Example `BC Flux for Energy on MESH_PART= distribution_factor multiplier=3.2`

Description Sets the surface flux to the value of the distribution factor scaled by the MULTIPLIER parameter. Here the distribution factor is present on a mesh entity associated with a sideset (surface) of named *MESH_PART*.

9.2.4 BC FLUX = ENCLOSURE_RADIATION

Parameters `[MULTIPLIER = REAL]`

Example `BC Flux for Energy on surface_10 = Enclosure_Radiation`

Description This boundary condition incorporates the heat flux that's computed using Chaparral for enclosure radiation.

See chapter 22 for more information on enclosure radiation.

9.2.5 BC FLUX = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example `BC Flux for Energy on surface_10 = Encore_Function Name=My_Function`

Description Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument EVAL_TYPE can be used to select the particular function on the Encore function object. Valid options include VALUE, DOT, GRADIENT, FLUX and STRESS. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – VALUE for NO_OP expressions, DOT for DT_OP expressions and GRADIENT for GRAD_OP expressions.

9.2.6 BC FLUX = PID_CONTROLLED

Parameters

```
Max_Output = REAL
Setpoint_Function_Name = STRING
Control_Variable = STRING
[P = REAL]
[I = REAL]
[D = REAL]
[Band = REAL]
[Filter_Tau = REAL]
```

Example

```
BC Flux for Energy on surface_1 = PID_Controlled P=0.1 I=1e-4
Max_Output=20000 Setpoint_Function_Name=pid_setpoints1
Control_Variable=ctrlT1
```

Description

This boundary condition imposes an energy flux using a PID controller. The controller setpoint vs. time is defined in a user function, and the controller feedback point uses a global variable (a data probe or Encore function).

The derivative term uses a filtered error value, using a low pass filter whose time constant is specified by the user.

Like a heater, this is only capable of providing a positive influx. Even if the controller output is negative, the applied heat flux will be truncated at 0. Likewise, the applied heat flux cannot exceed the specified max flux.

The nominal scaled PID controller output (from 0 to 1) is

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{\partial e_f}{\partial t} \tag{9.15}$$

where it is important to note that the K_p coefficient does not scale the others. In some PID implementations, the K_i and K_d coefficients are internally multiplied by K_p so care must be taken to provide the correct coefficients. This output from 0 to 1 is scaled by the specified Max_Output value to produce the applied energy flux. This means you should not include the output magnitude scaling in your P, I, or D parameters.

The P, I, and D coefficients can be specified in a number of ways:

- Just give a 'Band' argument. This calculates P = 1/Band and uses the defaults of I = 0.01*P and D = 0.
- Give a 'Band' argument and non-defaults for I and/or D. This still multiplies whatever you give for I and D by P.
- Give a 'P' argument instead of 'Band'. NOTE: in this mode the I and D values are not scaled by P.

If you provide both P and Band, P is used, the Band argument is ignored, and I and D are not scaled.

The default values for I and D are 0.01 and 0 if not provided.

The setpoint, controller output, error, filtered error, and filtered error time derivative are all output to automatically created global variables for diagnostics and postprocessing.

9.2.7 BC FLUX = PID_CONTROLLED_COOLER

Parameters
```
Max_Cooling_Power = REAL
Setpoint_Function_Name = STRING
Control_Variable = STRING
[P = REAL]
[I = REAL]
[D = REAL]
[Band = REAL]
[Filter_Tau = REAL]
```

Example
```
BC Flux for Energy on surface_1 = PID_Controlled_Cooler P=0.1 I=1e-4
Max_Cooling_Power=20000 Setpoint_Function_Name=pid_setpoints1
Control_Variable=ctrlT1
```

Description This is similar to PID_Controlled, however, the flux is applied to cool the surface instead of heat it. Max_Cooling_Power should be a positive value.

9.2.8 BC FLUX = PID_CONTROLLED_BIDIRECTIONAL

Parameters
```
Max_Cooling_Power = REAL
Max_Heating_Power = REAL
Setpoint_Function_Name = STRING
Control_Variable = STRING
[P = REAL]
[I = REAL]
[D = REAL]
[Band = REAL]
[Filter_Tau = REAL]
```

Example
```
BC Flux for Energy on surface_1 = PID_Controlled_Bidirectional
P=0.1 I=1e-4 Max_Cooling_Power=20000 Max_Heating_Power=20000
Setpoint_Function_Name=pid_setpoints1 Control_Variable=ctrlT1
```

Description This is similar to PID_Controlled, however, the flux is applied to cool and heat the surface instead of just heating it. Max_Cooling_Power and Max_Heating_Power should both be positive values.

9.2.9 BC FLUX = LASER

Parameters
```
T_ON = REAL
T_OFF = REAL
X_0 = REAL
Y_0 = REAL
R0 = REALW = REAL
FLUX = REAL
R = REAL
```

Example

```
BC Flux for Energy on surface_10 = Laser t_on=0 t_off=10 X_0=0 Y_0=0 w=1.0
r=0.025 flux=3.14159 r0=0.01
```

Description

This boundary condition imposes an energy flux due to an incident laser. The laser is directed in the z direction and travels in a circular path of radius r0.

`T_ON` is the time when the laser is turned on.

`T_OFF` is the time when the laser is turned off.

`X_0` is the x coordinate of the center of the laser path.

`Y_0` is the y coordinate of the center of the laser path.

`R0` is the radius of the circular laser path.

`W` is the angular velocity of the laser.

`FLUX` is the value of the energy flux *into* the surface. (The usual convention is that positive fluxes indicate loss.)

`R` is the radius of the laser beam.

9.2.10 BC FLUX = LASER_WELD

Parameters

```
PATH_FUNCTION = STRING
FLUX = REAL
R = REAL
[NORMAL_TOLERANCE = REAL]
```

Example

```
# This function lives at the top level, inside the 'Begin Sierra' block


Begin Definition for Function PATH
  Type is Multicolumn Piecewise Linear
  Column Titles Time X Y Z
  Begin Values
    0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00
    0.2250000E+00   0.1500000E+00   0.0000000E+00   0.0000000E+00
    0.2328540E+00   0.1552336E+00  -0.1370490E-03   0.0000000E+00
    ...
   End
End



# This goes in the Aria Region with all of the other BCs
BC Flux for Energy on surface_10 = Laser Path_Function=PATH R=0.025
Flux=3.14159
```

Description

This boundary condition imposes an energy flux due to an incident laser. The position of the laser is dictated by a user supplied function which contains x, y and z coordinates as a function of time; Aria uses linear piecewise interpolation to obtain the location of the laser at a given time.

`PATH` is the name of the user supplied function which has time as the first column and contains columns titled `X`, `Y` and `Z` containing the x, y and z coordinates.

`FLUX` is the value of the energy flux *into* the surface. (The usual convention is that positive fluxes indicate loss.)

R is the radius of the laser beam.

The determination of whether a point in space has an incident laser flux is done as follows. At time t the laser is at a point $\boldsymbol{P}$ and the Laser_Weld boundary condition is being evaluated at a point $\boldsymbol{x}$ (typically an integration point). The distance of $\boldsymbol{x}$ from $\boldsymbol{P}$ is defined as $\boldsymbol{r} \equiv \boldsymbol{P} - \boldsymbol{x}$. If a tolerance NORMAL_TOLERANCE is supplied and the out-of-surface portion of $\boldsymbol{r}$ is greater than the tolerance then the flux is taken to be zero. Otherwise, the out-of-surface portion of $\boldsymbol{r}$ is subtracted to neglect minor differences between the PATH_FUNCTION and the discretized mesh, $\boldsymbol{r}_s \equiv \boldsymbol{r} - (\boldsymbol{n} \cdot \boldsymbol{r})\boldsymbol{n}$ where $\boldsymbol{n}$ is the outward unit normal at $\boldsymbol{x}$. If $|\boldsymbol{r}_s| <$ R then the point $\boldsymbol{x}$ is considered to be in the beam and the flux is applied; otherwise, no flux is applied.

9.2.11 BC FLUX = GAUSSIAN_LINE_WELD

Parameters

```
SRC_X = REAL
SRC_Y = REAL
[SRC_Z = REAL]
DIR_X = REAL
DIR_Y = REAL
[DIR_Z = REAL]
VEL_X = REAL
VEL_Y = REAL
[VEL_Z = REAL]
FLUX = REAL
R = REAL
R_EFF = REAL
[T_ON = REAL]
[T_OFF = REAL]
[ALPHA = REAL]
[COMPUTE_VISIBILITY_FIELD = STRING]
[DEPTH_ENHANCED_LASER_ABSORPTION_COEFFICIENT = STRING]
```

Example

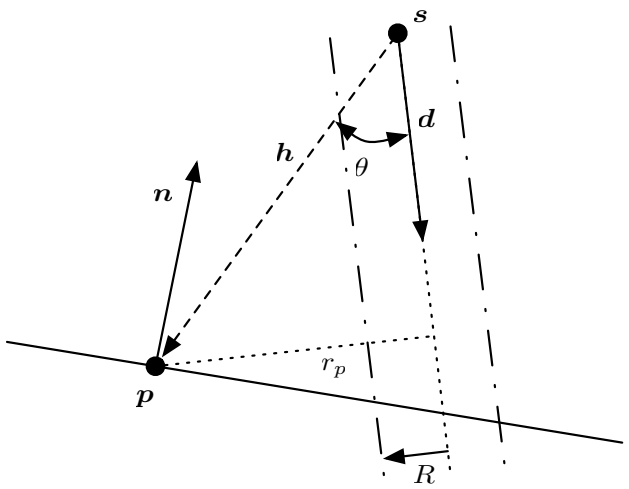
```
BC Flux for Energy on surface_10 = Gaussian_Line_Weld Src_X=0 Src_Y=3
Dir_X=0 Dir_Y=-1 Flux=1000 R=0.01 VEL_X=0.1 VEL_Y=0
```

Description

This boundary condition imposes an energy flux due to an incident laser. The laser source is at the coordinates

$$\boldsymbol{s} = \boldsymbol{x}_s + \boldsymbol{v}_s(t - t_{on}) \tag{9.16}$$

where $\boldsymbol{x}_s$ is provided by the SRC vector coordinates and $\boldsymbol{v}_s$ is provided by the VEL vector coordinates. The laser is directed from $\boldsymbol{s}$ by the DIR vector ($\boldsymbol{d}$ in the figure below).



The point-wise energy flux, f_{eff}, is computed using a Gaussian distribution based on the effective radius R_{eff} (`R_EFF`),

$$f_{eff} = 2R_{eff} f e^{\frac{-R_{eff} r_p^2}{R^2}} \tag{9.17}$$

where r_p is the radial coordinate from the center of the incident laser.

The radius of the laser is provided by `R` (R in the figure). A positive `FLUX` is defined as energy input to the surface. That is, the net normal flux is

$$q_n = \frac{(-\boldsymbol{n}) \cdot \boldsymbol{d}}{|\boldsymbol{d}|}(-f_{eff}) = \frac{\boldsymbol{n} \cdot \boldsymbol{d}}{|\boldsymbol{d}|} f_{eff} \tag{9.18}$$

Here, the term $-\boldsymbol{n} \cdot \boldsymbol{d}$ accounts for the fact that the surface may not be orthogonal to the laser.

The optional parameter `ALPHA` is the absorption coefficient for the incident laser flux (defaults to 1.0).

The optional parameter `COMPUTE_VISIBILITY_FIELD` is used to enable the calculation of the visibility of the laser to the given surface. When this parameter is given, the provided field will be populated with 1 for each node of the surface that is visible to the laser, and 0 for each node that is obstructed by other portions of the surface. This field is then used to multiply the incident flux so that the obstructed portions of the surface do not get heated by the laser.

The optional parameter `DEPTH_ENHANCED_LASER_ABSORPTION_COEFFICIENT` is used to specify a filename containing tabular data that will be used to compute an absorption coefficient that depends on the relative depth of the point and the aspect ratio of the given surface, defined as the surface depth/`R`. The text file is formatted as follows. The first line contains a list of the relative depth values and the second line contains of list of aspect ratio values. It is assumed that the number of relative depth values and aspect ratios are the same. The remaining rows define a table of absorption coefficient for increasing depth values and fixed increasing aspect ratios. Bilinear interpolation is used to evaluated the enhanced absorption coefficient for each point on the surface. For any point that lies outside of the specified range of relative depth or aspect ratio values, the value will be extrapolated using the nearest bilinear cell.

The net absorption coefficient is the product of `ALPHA` (defaults to 1.0), the results of the optional `COMPUTE_VISIBILITY_FIELD` calculation, and the optional `DEPTH_ENHANCED_LASER_ABSORPTION_COEFFI` calculation.

9.2.12 BC FLUX = GAUSSIAN_SPOT_WELD

Parameters

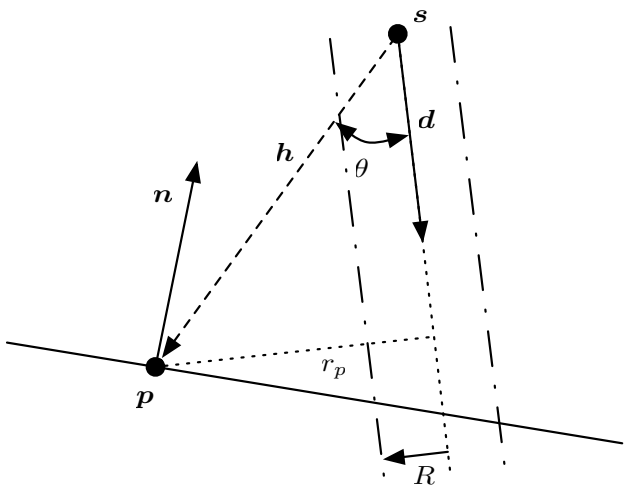
```
SRC_X = REAL
SRC_Y = REAL
[SRC_Z = REAL]
DIR_X = REAL
DIR_Y = REAL
[DIR_Z = REAL]
FLUX = REAL
R = REAL
R_EFF = REAL
[T_ON = REAL]
[T_OFF = REAL]
[ALPHA = REAL]
[COMPUTE_VISIBILITY_FIELD = STRING]
[DEPTH_ENHANCED_LASER_ABSORPTION_COEFFICIENT = STRING]
```

Example

```
BC Flux for Energy on surface_10 = Gaussian_Spot_Weld Src_X=0 Src_Y=3
Dir_X=0 Dir_Y=-1 Flux=1000 R=0.01
```

Description

This boundary condition imposes an energy flux due to an incident laser. The laser source is at the coordinates provided by the SRC vector coordinates ($\boldsymbol{s}$ in the figure below) and it is directed from there as specified by the DIR vector ($\boldsymbol{d}$ in the figure below).



The point-wise energy flux, f_{eff}, is computed using a Gaussian distribution based on the effective radius R_{eff} (R_EFF),

$$f_{eff} = 2R_{eff} f e^{\frac{-R_{eff} r_p^2}{R^2}} \tag{9.19}$$

where r_p is the radial coordinate from the center of the incident laser.

The radius of the laser is provided by R (R in the figure). A positive FLUX is defined as energy input to the surface. That is, the net normal flux is

$$q_n = \frac{(-\boldsymbol{n}) \cdot \boldsymbol{d}}{|\boldsymbol{d}|}(-f_{eff}) = \frac{\boldsymbol{n} \cdot \boldsymbol{d}}{|\boldsymbol{d}|} f_{eff} \tag{9.20}$$

Here, the term $-\boldsymbol{n} \cdot \boldsymbol{d}$ accounts for the fact that the surface may not be orthogonal to the laser.

The optional parameter ALPHA is the absorption coefficient for the incident laser flux (defaults to 1.0).

The optional parameter COMPUTE_VISIBILITY_FIELD is used to enable the calculation of the visibility of the laser to the given surface. When this parameter is given, the provided field will be populated with 1 for each node of the surface that is visible to the laser, and 0 for each node that is obstructed by other portions of the surface. This field is then used to multiply the incident flux so that the obstructed portions of the surface do not get heated by the laser.

The optional parameter DEPTH_ENHANCED_LASER_ABSORPTION_COEFFICIENT is used to specify a filename containing tabular data that will be used to compute an absorption coefficient that depends on the relative depth of the point and the aspect ratio of the given surface, defined as the surface depth/R. The text file is formatted as follows. The first line contains a list of the relative depth values and the second line contains of list of aspect ratio values. It is assumed that the number of relative depth values and aspect ratios are the same. The remaining rows define a table of absorption coefficient for increasing depth values and fixed increasing aspect ratios. Bilinear interpolation is used to evaluated the enhanced absorption coefficient for each point on the surface. For any point that lies outside of the specified range of relative depth or aspect ratio values, the value will be extrapolated using the nearest bilinear cell.

The net absorption coefficient is the product of ALPHA (defaults to 1.0), the results of the optional COMPUTE_VISIBILITY_FIELD calculation, and the optional DEPTH_ENHANCED_LASER_ABSORPTION_COEFF calculation.

9.2.13 BC FLUX = GENERALIZED_NAT_CONV

Parameters `[MULTIPLIER = REAL]`

Example `BC Flux for Energy on surface_10 = Generalized_Nat_Conv`

Description This boundary condition adds a heat flux due to Newton's law of cooling. This is similar to the NAT_CONV boundary condition except that it utilizes the HEAT_TRANSFER_COEFFICIENT and BC_REFERENCE_TEMPERATURE models.

$$q = mh(T - T_{ref}) \tag{9.21}$$

where m is the optional MULTIPLIER parameter (defaults to 1.0), h is the heat transfer coefficient, T is the temperature at the surface and T_{ref} is a reference temperature.

9.2.14 BC FLUX = GENERALIZED_RAD

Parameters `[MULTIPLIER = REAL]`

Example `BC Flux for Energy on surface_10 = Generalized_Rad`

Description This boundary condition accounts for the heat flux due to radiation in free space, similar to the RAD boundary condition but the model is more general in that it uses specific models for

EMISSIVITY, RADIATION FORM FACTOR and BC RAD REFERENCE TEMPERATURE. The functional form of this boundary condition is

$$q = mF\sigma\epsilon(T^4 - T_{ref}^4) \tag{9.22}$$

where m is the optional MULTIPLIER parameter (defaults to 1.0), F is the RADIATION FORM FACTOR, σ is the Stefan-Boltzmann constant, ϵ is the emissivity of the surface, T is the temperature at the surface and T_{ref} is a reference temperature which must be supplied from a BC RAD REFERENCE TEMPERATURE material model.

The Stefan-Boltzmann constant must be defined in the input file using a Global Constants block, e.g.,

```
Begin Global Constants
   Stefan Boltzmann Constant = 5.67e-14
End
```

9.2.15 BC FLUX = SHARP_LINE_WELD

Parameters

```
SRC_X = REAL
SRC_Y = REAL
[SRC_Z = REAL]
DIR_X = REAL
DIR_Y = REAL
[DIR_Z = REAL]
VEL_X = REAL
VEL_Y = REAL
[VEL_Z = REAL]
FLUX = REAL
R = REAL
[T_ON = REAL]
[T_OFF = REAL]
[ALPHA = REAL]
[COMPUTE_VISIBILITY_FIELD = STRING]
[DEPTH_ENHANCED_LASER_ABSORPTION_COEFFICIENT = STRING]
```

Example

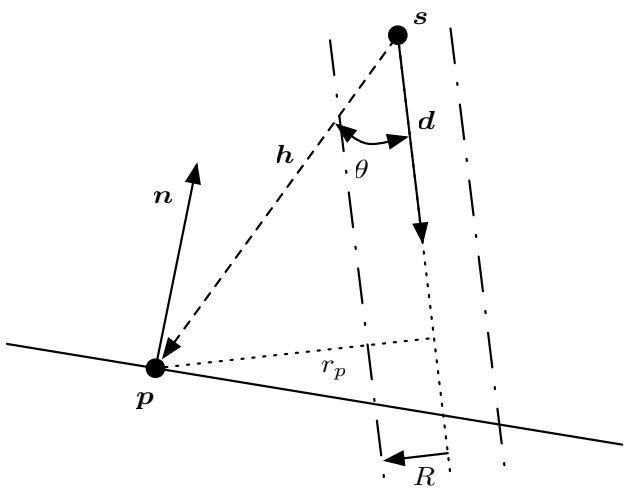
```
BC Flux for Energy on surface_10 = Sharp_Line_Weld Src_X=0 Src_Y=3 Dir_X=0
Dir_Y=-1 Flux=1000 R=0.01 VEL_X=0.1 VEL_Y=0
```

Description

This boundary condition imposes an energy flux due to an incident laser. The laser source is at the coordinates

$$\boldsymbol{s} = \boldsymbol{x}_s + \boldsymbol{v}_s(t - t_{on}) \tag{9.23}$$

where $\boldsymbol{x}_s$ is provided by the SRC vector coordinates and $\boldsymbol{v}_s$ is provided by the VEL vector coordinates. The laser is directed from $\boldsymbol{s}$ by the DIR vector ($\boldsymbol{d}$ in the figure below).



The radius of the laser is provided by `R` (R in the figure) and the energy flux is provided by the `FLUX` argument (written as f below). A positive `FLUX` is defined as energy input to the surface. That is, the net normal flux is

$$q_n = \frac{(-\boldsymbol{n}) \cdot \boldsymbol{d}}{|\boldsymbol{d}|}(-f) = \frac{\boldsymbol{n} \cdot \boldsymbol{d}}{|\boldsymbol{d}|} f \tag{9.24}$$

Here, the term $-\boldsymbol{n} \cdot \boldsymbol{d}$ accounts for the fact that the surface may not be orthogonal to the laser.

The optional parameter `ALPHA` is the absorption coefficient for the incident laser flux (defaults to 1.0).

The optional parameter `COMPUTE_VISIBILITY_FIELD` is used to enable the calculation of the visibility of the laser to the given surface. When this parameter is given, the provided field will be populated with 1 for each node of the surface that is visible to the laser, and 0 for each node that is obstructed by other portions of the surface. This field is then used to multiply the incident flux so that the obstructed portions of the surface do not get heated by the laser.

The optional parameter `DEPTH_ENHANCED_LASER_ABSORPTION_COEFFICIENT` is used to specify a filename containing tabular data that will be used to compute an absorption coefficient that depends on the relative depth of the point and the aspect ratio of the given surface, defined as the surface depth/`R`. The text file is formatted as follows. The first line contains a list of the relative depth values and the second line contains of list of aspect ratio values. It is assumed that the number of relative depth values and aspect ratios are the same. The remaining rows define a table of absorption coefficient for increasing depth values and fixed increasing aspect ratios. Bilinear interpolation is used to evaluated the enhanced absorption coefficient for each point on the surface. For any point that lies outside of the specified range of relative depth or aspect ratio values, the value will be extrapolated using the nearest bilinear cell.

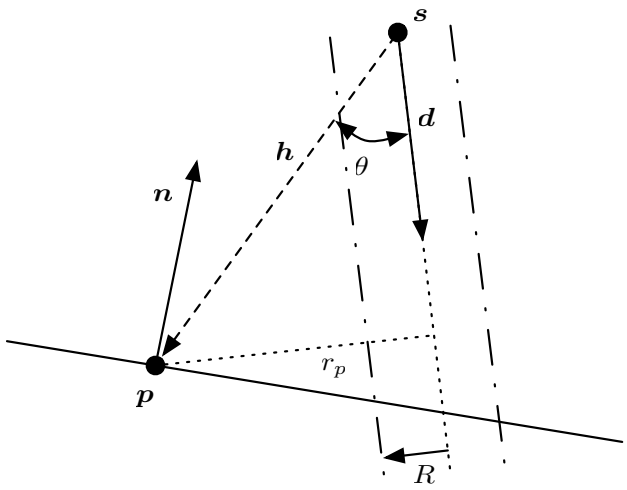
The net absorption coefficient is the product of `ALPHA` (defaults to 1.0), the results of the optional `COMPUTE_VISIBILITY_FIELD` calculation, and the optional `DEPTH_ENHANCED_LASER_ABSORPTION_COEFFI` calculation.

9.2.16 BC FLUX = SHARP_SPOT_WELD

Parameters
```
SRC_X = REAL
SRC_Y = REAL
[SRC_Z = REAL]
DIR_X = REAL
DIR_Y = REAL
[DIR_Z = REAL]
FLUX = REAL
R = REAL
[T_ON = REAL]
[T_OFF = REAL]
```

Example
```
BC Flux for Energy on surface_10 = Sharp_Spot_Weld Src_X=0 Src_Y=3 Dir_X=0
Dir_Y=-1 Flux=1000 R=0.01
```

Description This boundary condition imposes an energy flux due to an incident laser. The laser source is at the coordinates provided by the SRC vector coordinates ($\boldsymbol{s}$ in the figure below) and it is directed from there as specified by the DIR vector ($\boldsymbol{d}$ in the figure below).



The radius of the laser is provided by R (R in the figure) and the energy flux is provided by the FLUX argument (written as f below). A positive FLUX is defined as energy input to the surface. That is, the net normal flux is

$$q_n = \frac{(-\boldsymbol{n})\cdot\boldsymbol{d}}{|\boldsymbol{d}|}(-f) = \frac{\boldsymbol{n}\cdot\boldsymbol{d}}{|\boldsymbol{d}|}f \tag{9.25}$$

Here, the term $-\boldsymbol{n}\cdot\boldsymbol{d}$ accounts for the fact that the surface may not be orthogonal to the laser.

9.2.17 BC FLUX = LATENT_HEAT

Parameters
```
Y0 = REAL
[SPECIES = INT]
```

Example
```
BC Flux for Energy on surface_10 = Latent_Heat Y0=0.2 SPECIES=0
BC Flux for Energy on surface_10 = Latent_Heat Y0=0.4 SPECIES=1
BC Flux for Energy on surface_10 = Latent_Heat Y0=0.0 SPECIES=2
```

Description This boundary condition accounts for the heat flux due to the latent heat of vaporization (evaporation).

$$q = H_v \rho (Y_i - Y_{\infty,i}) \tag{9.26}$$

where $H_{v,i}$ is the heat of vaporization of species i, ρ is the density of the material, Y_i is the mass fraction of species i and $Y_{\infty,i}$ is the mass fraction of species i far from the surface.

`SPECIES` is the index of the species to use (in multicomponent systems), $= i$ in equation 9.26.

`Yinf` is the mass fraction of species i far from the surface, $= Y_{\infty,i}$ in equation 9.26.

9.2.18 BC FLUX = NAT_CONV

Parameters
```
T_REF = REAL
H = REAL
```

Example
```
BC Flux for Energy on surface_10 = Nat_Conv T_REF=273.15 H=300
```

Description This boundary condition accounts for the heat flux due to natural heat convection:

$$q = h(T - T_{ref}) \tag{9.27}$$

`T_REF` is the reference temperature of free space.

`H` is the heat transfer coefficient.

9.2.19 BC FLUX = RAD

Parameters
```
T_REF = REAL
CRAD = REAL
```

Example
```
BC Flux for Energy on surface_10 = Rad T_REF=273.15 CRAD=1.3e-8
```

Description This boundary condition accounts for the heat flux due to radiation in free space:

$$q = c_{rad}(T^4 - T_{ref}^4) \tag{9.28}$$

`T_REF` is the temperature of free space.

`CRAD` is the coefficient that multiplies the radiation term, viz., the product of the emissivity and the Stefan-Boltzmann constant, which is $5.67 \times 10^{-08}\,\mathrm{W\,m^{-2}\,K^{-4}}$ (SI Units).

9.2.20 BC FLUX = VAPOR_COOLING

Parameters
```
TBOIL = REAL
```

Example
```
BC Flux for Energy on surface_10 = Vapor_Cooling Tboil=3000
```

Description This boundary condition accounts for the cooling of a material due to vaporization.

`TBOIL` is the boiling point of the material.

9.2.21 BC FLUX = NAT_CONV

Parameters
```
Yinf = REAL
k = REAL
```

Example
```
BC Flux for Species_2 on surface_10 = Nat_Conv k=0.15 Yinf=0.8
```

Description This boundary condition accounts for the mass flux due to natural convection:

$$q = k(Y - Y_\infty) \tag{9.29}$$

`Yinf` is the bulk species concentration, Y_∞.

`k` is the mass transfer coefficient.

9.2.22 BC FLUX = CAPILLARY

Parameters
```
(none)
```

Example
```
BC FluxBP for Momentum on surface_10 = Capillary
```

Description This boundary condition implements the capillary (surface tension) contributions to the traction boundary condition [19]. Specifically, this boundary condition adds to the k^{th} component of the i^{th} momentum residual

$$\int_{\text{S}} \sigma \left(\boldsymbol{I} - \boldsymbol{nn} \right) : \boldsymbol{\nabla} \left(\phi^i \boldsymbol{e}^k \right) \, \text{dS}. \tag{9.30}$$

where σ is the surface tension and $\boldsymbol{n}$ is the unit outward normal to the interface. This condition accounts for both the curvature and surface tension gradient contributions to the traction condition.

9.2.23 BC FLUX = CONSTANT_TRACTION

Parameters
```
[X = REAL]
[Y = REAL]
[Z = REAL]
```

Example
```
BC Flux for Momentum on surface_10 = Constant_Traction Y=0.5
or
BC Flux for Mesh on surface_10 = Constant_Traction X=0.1 Y=0.5
or
BC Flux for Solid on surface_10 = Constant_Traction Z=0.5
```

Description This boundary condition integrates a constant and uniform traction over a surface for either fluid momentum, mesh elasticity or solid elasticity. Specifically, this boundary condition adds to the k^{th} component of the i^{th} momentum/mesh/solid residual

$$\int_{\text{S}} \boldsymbol{f}_t \phi^i \, \text{dS}. \tag{9.31}$$

where $\boldsymbol{f}_t$ is the constant traction vector whose components are given by the parameters `X`, `Y` and `Z`.

9.2.24 BC FLUX = ELECTRIC_TRACTION

Parameters `[SIGN = REAL]`

Example
```
BC Flux for Momentum on surface_10 = Electric_Traction
BC Flux for Mesh on surface_10 = Electric_Traction
BC Flux for Solid on surface_10 = Electric_Traction
```

Description This boundary condition adds the stress contribution for due to the presence of an electric field. Here, the electric stress tensor is taken to be

$$\boldsymbol{T}_e = \epsilon \boldsymbol{E}\boldsymbol{E} - \frac{1}{2}\epsilon \boldsymbol{E}\cdot\boldsymbol{E}\boldsymbol{I} \tag{9.32}$$

where $\boldsymbol{E} = -\boldsymbol{\nabla} V$ is the electric field and V is voltage and ϵ is the electric permittivity. The electric traction is then defined as

$$\boldsymbol{t} = \epsilon \boldsymbol{n}\cdot\boldsymbol{E}\boldsymbol{E} - \frac{1}{2}\epsilon \boldsymbol{E}\cdot\boldsymbol{E}\boldsymbol{n} \tag{9.33}$$

where $\boldsymbol{n}$ is the outward normal to the boundary.

9.2.25 BC FLUX = NAT_E

Example
```
BC Flux for Charge_Density on surface_1 = Nat_E
```

Description This boundary condition creates a no-flux boundary condition for the charge density equation when DIFF is activated in the equation. The diffusion operator for this equation is not of the standard no-flux type, and so this term needs to be added to get the correct surface term contribution.

9.2.26 BC FLUX = FLOW_HYDROSTATIC

Parameters
```
[GX = REAL]
[GY = REAL]
[GZ = REAL]
[P_REF = REAL]
```

Example
```
BC Flux for Momentum on surface_10 = Flow_Hydrostatic Gy=-9.8
```

Description This boundary condition provides a hydrostatic pressure head along a boundary. The acceleration vector g is specified with the parameters `GX`, `GY` and `GZ` (all default to zero). The reference pressure P_{ref} (`P_REF`, defaults to zero) is taken to be the pressure datum at the origin $(0,0,0)$. Specifically, this BC adds

$$\boldsymbol{n}\cdot\boldsymbol{T} = -\left(P_{ref} + \sum_i^{N_d} \rho g_i x_i\right)\boldsymbol{n} \tag{9.34}$$

to the momentum equation. Here N_d is the spatial dimension of the problem, g_i are the components of the acceleration, x_i are the coordinate components and ρ is the density. **NOTE:** This BC evaluates the density material model for ρ thus the parameters for $\boldsymbol{g}$ should *not* include the density.

9.2.27 BC FLUX = FREE_OPEN_FLOW

Parameters
```
PRESSURE = REAL
```

Example
```
BC Flux for Momentum on surface_10 = Free_Open_Flow Pressure=1000
```

Description This boundary condition adds back the stress contribution for inlet/outlet flows where the pressure is prescribed. This command line is used in open-flow applications to set the pressure datum but imposes no other constraints on the flow profile.

$$\boldsymbol{q} = -p_\circ \boldsymbol{n} + \mu \boldsymbol{n} \cdot \left(\boldsymbol{\nabla v} + \boldsymbol{\nabla v}^t \right) \tag{9.35}$$

Here, $p_\circ$ is the pressure provided by the user, μ is the viscosity, $\boldsymbol{\nabla v}$ is the velocity gradient, and $\boldsymbol{n}$ is the outward normal to the boundary.

9.2.28 BC FLUX = OPEN_FLOW

Parameters
```
PRESSURE = REAL
```

Example
```
BC Flux for Momentum on surface_10 = Open_Flow Pressure=1000
```

Description This boundary condition adds back the stress contribution for inlet/outlet flows where the pressure is prescribed, and the flow is unidirectional and fully developed. Specifically, it removes the normal component of the viscous stresses (and any other non-pressure stresses).

$$\boldsymbol{q} = -p_\circ \boldsymbol{n} + \boldsymbol{n} \cdot \boldsymbol{\tau} \cdot (\boldsymbol{I} - \boldsymbol{nn}) \tag{9.36}$$

Here, $p_\circ$ is the pressure provided by the user $\boldsymbol{n}$ is the outward normal to the boundary and $\boldsymbol{\tau}$ is the sum of all other stresses except the the isotropic pressure (e.g., `NEWTONIAN_PRESSURE_MOMENTUM_STRESS`)

9.2.29 BC FLUX = PRESSURE

Parameters
```
P = REAL
[C_T = REAL]
```

Example
```
BC Flux for Momentum on surface_10 = Pressure P=101325
or
BC Flux for Mesh on surface_10 = Pressure P=101325
or
BC Flux for Solid on surface_10 = Pressure P=101325
```

Description This boundary condition integrates a uniform pressure over a surface for either fluid momentum, mesh elasticity or solid elasticity. The optional parameter `C_T` allows the pressure to vary linearly in time. Specifically, this boundary condition adds to the k^{th} component of the i^{th} momentum/mesh/solid residual

$$\int_{\text{S}} -\left(p + c_t t\right) \boldsymbol{n}\phi^i \, \text{dS}. \tag{9.37}$$

where p is the pressure provided by the parameter `P`, t is time and c_t is a constant provided by the `C_T` parameter.

9.2.30 BC FLUX = LUBRICATION_PRESSURE

Parameters `(none)`

Example
```
BC Flux for Mesh on surface_10 = Lubrication_Pressure
or
BC Flux for Solid on surface_10 = Lubrication_Pressure
```

Description This boundary condition integrates the lubrication pressure (the DOF from the lubrication equation) over a surface. This is intended to apply the fluid pressure to a solid region, and can be used for either the MESH or SOLID equations.

9.2.31 BC FLUX = ORIENTED_SLIP

Parameters
```
BETA_NORMAL = REAL
BETA_TANGENT = REAL
[VS_X = REAL]
[VS_Y = REAL]
[VS_Z = REAL]
```

Example
```
BC Flux for Momentum on surface_7 = Oriented_Slip Beta_Tangent=1e-1
Beta_Normal=1e-5
```

Description This is similar to the classic `SLIP` condition except that it permits separate slip coefficients for the normal and tangent directions.

$$\boldsymbol{q} = \boldsymbol{n} \cdot \boldsymbol{T} = \frac{1}{\beta_t}\left(\boldsymbol{v}_s - \boldsymbol{v}\right) + \left(\frac{1}{\beta_n} - \frac{1}{\beta_t}\right)\left(\boldsymbol{v}_s - \boldsymbol{v}\right) \cdot \boldsymbol{nn} \tag{9.38}$$

where β_t is the Navier slip coefficient for the tangent direction (`BETA_TANGENT`), β_n is that for the normal direction (`BETA_NORMAL`, $\boldsymbol{v}$ is the fluid velocity and $\boldsymbol{v}_s$ is the velocity of the surface. The surface velocity is zero by default but a nonzero velocity can be supplied in component form using one or more of the optional `VS_X`, `VS_Y` and `VS_Z` parameters.

9.2.32 BC FLUX = SLIP

Parameters
```
BETA = REAL
[VS_X = REAL]
[VS_Y = REAL]
[VS_Z = REAL]
```

Example `BC Flux for Momentum on surface_10 = Slip Beta = 0.01`

Description This boundary condition implements the Navier slip boundary condition along a surface wherein the tangential velocity along the surface is proportional to the fluid stress,

$$\boldsymbol{q} = \boldsymbol{n} \cdot \boldsymbol{T} = \frac{1}{\beta} (\boldsymbol{v}_s - \boldsymbol{v}) \tag{9.39}$$

where β is the Navier slip coefficient, $\boldsymbol{v}$ is the fluid velocity and $\boldsymbol{v}_s$ is the velocity of the surface. The surface velocity is zero by default but a nonzero velocity can be supplied in component form using one or more of the optional `VS_X`, `VS_Y` and `VS_Z` parameters.

9.2.33 BC FLUX = TRANSIENT_TRACTION

Parameters
```
[A_X = REAL]
[A_Y = REAL]
[A_Z = REAL]
[B_X = REAL]
[B_Y = REAL]
[B_Z = REAL]
```

Example
```
BC Flux for Momentum on surface_10 = Transient_Traction B_Y=0.5
or
BC Flux for Mesh on surface_10 = Transient_Traction A_X=0.1 A_Y=0.5
or
BC Flux for Solid on surface_10 = Transient_Traction B_Z=0.5
```

Description This boundary condition integrates a uniform but time dependent traction over a surface for either fluid momentum, mesh elasticity or solid elasticity. Specifically, this boundary condition adds to the k^{th} component of the i^{th} momentum/mesh/solid residual

$$\int_{\text{S}} (\boldsymbol{f}_a + t\boldsymbol{f}_b)\, \phi^i \, \text{dS}. \tag{9.40}$$

where t is time and $\boldsymbol{f}_a$ and $\boldsymbol{f}_b$ are is constant traction vectors whose components are given by the parameters `A_X`, `A_Y` and `A_Z` and `B_X`, `B_Y` and `B_Z` respectively.

9.2.34 BC FLUX = WETTING_SPEED_BLAKE_LS

Parameters
```
V_W = REAL
G = REAL
THETA = REAL
WIDTH = REAL
```

Example
```
BC Disting for Momentum_A on surface_3 = Wetting_Speed_Blake_LS V_w=1e-1 g=1
Width=1 Theta=60
BC Disting for Momentum_B on surface_3 = Wetting_Speed_Blake_LS V_w=1e-1 g=1
Width=1 Theta=60
```

Description This boundary condition is a distinguishing condition that enforces a slip velocity in accordance with the model provided by [20].

$$\boldsymbol{v} - f(\phi) v_w \sinh\left(g\left(\cos\theta_s - \cos\theta_a\right)\right) \frac{\boldsymbol{n}_{ls} - \cos\theta_a \boldsymbol{n}_w}{1 - \cos^2\theta_a} = \boldsymbol{0} \tag{9.41}$$

Here θ_s is the static or equilibrium contact angle and is provided by the `THETA` input parameter, θ_a is the actual (or "observed" or "current") contact angle, g is a constant parameter given by the `G` input parameter and v_w is the characteristic slip velocity and is given by the input parameter `V_W`. In their paper, Blake and De Coninck develop g and v_w theoretically.

The function $f(\phi)$ where ϕ is the level set distance function is simply a triangle shape function which causes the velocity to vary from v_w to zero over the distance given by the input parameter `WIDTH`. Taking h_w to half of the input `WIDTH` parameter, $f(\phi)$ is given as

$$f(\phi) = \begin{cases} 0 & : & & \phi & < & -h_w \\ 1 + \phi/h_w & : & -h_w & \leq \phi & < & 0 \\ 1 - \phi/h_w & : & 0 & \leq \phi & < & h_w \\ 0 & : & h_w & \leq \phi & & \end{cases}$$

This boundary condition is a distinguishing condition which means the momentum equations are discarded at the nodes where this is applied. Also, the velocity provided by this boundary condition is purely tangential. Thus, this boundary condition automatically enforces no-penetration in addition to slip/no-slip.

Note: The interface normal points out of the *negative* phase (negatively signed level set distance ϕ) into the *positive* phase. Thus, the contact angle is measured from the wall, through the *negative* phase and to the level set interface $\phi = 0$.

9.2.35 BC FLUX = USER_FUNCTION

Parameters
```
NAME = STRING
X = STRING
```

Example
```
BC Flux for Energy on surface_6 = USER_FUNCTION NAME=testflux X=TIME
```

Description The flux boundary condition value is defined by a tabular function of the given name which is evaluated as a function of argument X.

9.2.36 Nat Conv

Scope: Equation System

```
   Nat Conv Flux For Species [ {of|species|subindex} Species  |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent  |Opposing OpposingMeshExtent
]= Nat_Conv [ Using Data Specification Data Spec Name ][ K = k  |Yinf = yinf ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
k	real	undefined
yinf	real	undefined

Summary natconv species flux

9.2.37 Surface Stabilization

Scope: Equation System

Surface Stabilization Fluxbp For Species [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Surface_Stabilization [Using Data Specification *Data Spec Name*][D = *d*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
d	real	undefined

Summary surface diffusion term for species

9.2.38 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Species [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Far_Field_Entrainment_Value = *far_field_entrainment_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
far_field_entrainment_value	real	undefined

9.2.39 Fluid Solid Convection Coupled One Region

Scope: Equation System

Fluid Solid Convection Coupled One Region Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Fluid_Solid_Convection_Coupled_One_Region [Using Data Specification *Data Spec Name*][Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
multiplier	real	undefined

Summary Boundary condition for loose coupling of enthalpy to a free fluid region

9.2.40 Fluid Robin Coupled One Region

Scope: Equation System

Fluid Robin Coupled One Region Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Fluid_Robin_Coupled_One_Region [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Boundary condition for loose coupling of enthalpy to a free fluid region

9.2.41 Fluid Robin Coupled

Scope: Equation System

Fluid Robin Coupled Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Fluid_Robin_Coupled [Using Data Specification *Data Spec Name*][Enthalpy_Field = *enthalpy_field* |Enthalpy_Diffusive_Flux_Field = *enthalpy_diffusive_flux_field* |Coeff_Field = *coeff_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
enthalpy_field	"string"	undefined
enthalpy_diffusive_flux_field	"string"	undefined
coeff_field	"string"	undefined

Summary Boundary condition for loose coupling of enthalpy to a free fluid region

9.2.42 Generalized Nat Conv

Scope: Equation System

Generalized Nat Conv Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Generalized_Nat_Conv [Using Data Specification *Data Spec Name*][Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
multiplier	real	undefined

Summary Generalized natural convection flux BC

9.2.43 Fluid Robin Coupled With Solid Convection

Scope: Equation System

Fluid Robin Coupled With Solid Convection Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Fluid_Robin_Coupled_With_Solid_Convection [Using Data Specification *Data Spec Name*][Enthalpy_Field = *enthalpy_field* |Enthalpy_Diffusive_Flux_ = *enthalpy_diffusive_flux_field* |Coeff_Field = *coeff_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
enthalpy_field	"string"	undefined
enthalpy_diffusive_flux_field	"string"	undefined
coeff_field	"string"	undefined

Summary Boundary condition for loose coupling of enthalpy to a free fluid region

9.2.44 Fluid Solid Convection Coupled

Scope: Equation System

Fluid Solid Convection Coupled Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Fluid_Solid_Convection_Coupled [Using Data Specification *Data Spec Name*][Multiplier = *multiplier* |Fluid_Temperature_Field = *fluid_temperature_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
multiplier	real	undefined
fluid_temperature_field	"string"	undefined

Summary Boundary condition for loose coupling of enthalpy to a free fluid region

9.2.45 Convective Outflow

Scope: Equation System

Convective Outflow Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Convective_Outflow [Using Data Specification *Data Spec Name*][Ref_H = *ref_h*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
ref_h	real	undefined

9.2.46 User Vector Field

Scope: Equation System

User Vector Field Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= User_Vector_Field [Using Data Specification *Data Spec Name*][Name = *name* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
multiplier	real	undefined

Summary User vector field enthalpy flux

9.2.47 Open

Scope: Equation System

Open Flux For Porous_Enthalpy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open [Using Data Specification *Data Spec Name*][Pressure = *pressure* | Entrained_Value = *entrained_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
entrained_value	real	undefined

9.2.48 Wall Function

Scope: Equation System

Wall Function Disting For Cvfem_Turbulence_Dissipation_Rate [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshEx* | Opposing *OpposingMeshExtent*]= Wall_Function [Using Data Specification *Data Spec Name*][Wall_Friction_Factor = *Wall_Friction_Factor* | Wall_Velocity_X = *Wall_Velocity_X* | Wall_Velocity_Y = *Wall_Velocity_Y* | Wall_Velocity_Z = *Wall_Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Wall_Friction_Factor	real	undefined
Wall_Velocity_X	real	undefined
Wall_Velocity_Y	real	undefined
Wall_Velocity_Z	real	undefined

Summary CVFEM wall function turbulence dissipation rate distinguishing condition

9.2.49 Simple Interp

Scope: Equation System

Simple Interp Flux For Cvfem_Tke_Edge_Gradient_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM edge-based projection simple interpolation flux BC

9.2.50 Open Flow

Scope: Equation System

Open Flow Flux For Porous_Species [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.51 No Slip

Scope: Equation System

No Slip Disting For Solid [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= No_Slip [Using Data Specification *Data Spec Name*][VO = *v0*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined

Summary No slip model for SOLID distinguishing condition

9.2.52 Capillary

Scope: Equation System

```
Capillary Fluxbp For Cvfem_Momentum [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Capillary [ Using Data Specification Data Spec Name ][ ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC

9.2.53 Porous Robin One Region

Scope: Equation System

```
Porous Robin One Region Disting For Cvfem_Momentum [ {of|species|subindex} Species |{in
|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent
|Opposing OpposingMeshExtent ]= Porous_Robin_One_Region [ Using Data Specification Data Spec
Name ][ Alpha = alpha ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
alpha	real	undefined

Summary No slip model for momentum distinguishing condition

9.2.54 No Slip

Scope: Equation System

No Slip Disting For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= No_Slip [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary No slip model for momentum distinguishing condition

9.2.55 Porous Robin Interface

Scope: Equation System

Porous Robin Interface Disting For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Porous_Robin_Interface [Using Data Specification *Data Spec Name*][Alpha = *alpha* |Permeability_Field = *permeability_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
alpha	real	undefined
permeability_field	"string"	undefined

Summary No slip model for momentum distinguishing condition

9.2.56 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Cvfem_Mixture_Fraction [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure* |Far_Field_Entrainment_Value = *far_field_entrainment_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

9.2.57 Free Open Flow

Scope: Equation System

Free Open Flow Flux For Cvfem_Mixture_Fraction [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Free_Open_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.58 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Mixture_Fraction [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Total_Mdot = *total_mdot* |Inflow_Phi = *inflow_phi*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_mdot	real	undefined
inflow_phi	real	undefined

9.2.59 Wall Function

Scope: Equation System

Wall Function Disting For Hfem_Turbulent_Kinetic_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Wall_Function [Using Data Specification *Data Spec Name*][

Scale_Factor = *scale_factor* |Wall_Friction_Factor = *Wall_Friction_Factor* |Wall_Velocity_X = *Wall_Velocity_X* |Wall_Velocity_Y = *Wall_Velocity_Y* |Wall_Velocity_Z = *Wall_Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
scale_factor	real	undefined
Wall_Friction_Factor	real	undefined
Wall_Velocity_X	real	undefined
Wall_Velocity_Y	real	undefined
Wall_Velocity_Z	real	undefined

Summary HFEM wall function turbulent kinetic energy distinguishing condition

9.2.60 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Hfem_Specific_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExt* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure* |Far_Field_Entrainment_Value = *far_field_entrainm*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

Summary HFEM specific dissipation rate open advective flow flux BC

9.2.61 Kinematic

Scope: Equation System

Kinematic Disting For Solid_X [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for SOLID_x distinguishing condition

9.2.62 Pyrolysis

Scope: Equation System

```
Pyrolysis Disting For Solid_X [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Pyrolysis [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for SOLID_x distinguishing condition

9.2.63 Recession

Scope: Equation System

```
Recession Disting For Solid_X [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Recession [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for SOLID_x distinguishing condition

9.2.64 Capillary

Scope: Equation System

Capillary Fluxbp For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC

9.2.65 Wetting Outflow

Scope: Equation System

Wetting Outflow Edge For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]Where *SidePart0Name* Intersects *SidePart1Name* = Wetting_Outflow [Theta_S = *theta_s*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
SidePart0Name	string	undefined
SidePart1Name	string	undefined
theta_s	real	undefined

Summary Sharp wetting edge source term for outflow

9.2.66 Capillary Stabilization

Scope: Equation System

Capillary Stabilization Fluxbp For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Capillary_Stabilization [Using Data Specification *Data Spec Name*][Mu0 = *mu0* |Multiplier = *multiplier* |Use_Velocity_Correction = *use_velocity_correction*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC

9.2.67 Ls Capillary Stabilization

Scope: Equation System

Ls Capillary Stabilization Fluxbp For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Ls_Capillary_Stabilization [Using Data Specification *Data Spec Name*][Mu0 = *mu0* |Multiplier = *multiplier* |Use_Velocity_Correction = *use_velocity_correction*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC using level set normal

9.2.68 Fluid Robin Coupled One Region

Scope: Equation System

Fluid Robin Coupled One Region Flux For Mass_Balance [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExt* |Opposing *OpposingMeshExtent*]= Fluid_Robin_Coupled_One_Region [Using Data Specification *Data Spec Name*][Coeff_Scaling = *coeff_scaling*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
coeff_scaling	real	undefined

Summary Boundary condition for coupling of continuity to a free fluid in the same region.

9.2.69 Mass Fluid Robin Coupled

Scope: Equation System

Mass Fluid Robin Coupled Flux For Mass_Balance [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Fluid_Robin_Coupled [Using Data Specification *Data Spec Name*][Coeff_Scaling = *coeff_scaling* |P_Field = *P_Field* |Massflux_Field = *MassFlux_Field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
coeff_scaling	real	undefined
P_Field	"string"	undefined
MassFlux_Field	"string"	undefined

Summary Boundary condition for loose coupling of continuity to a free fluid region

9.2.70 Well Outflow

Scope: Equation System

Well Outflow Flux For Mass_Balance [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Well_Outflow [Using Data Specification *Data Spec Name*][Pi = *pi* | Pref = *pref*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pi	"string"	undefined
pref	"string"	undefined

9.2.71 Open

Scope: Equation System

Open Flux For Mass_Balance [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open [Using Data Specification *Data Spec Name*][Pressure = *pressure* | Entrained_Value = *entrained_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
entrained_value	real	undefined

9.2.72 Mass Fraction Fluid Robin Coupled One Region

Scope: Equation System

Mass Fraction Fluid Robin Coupled One Region Flux For Mass_Balance [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Mass_Fraction_Fluid_Robin_Coupled_One_Region [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Boundary condition for coupling of scalars to a free fluid in the same region.

9.2.73 Mass Fraction Fluid Robin Coupled

Scope: Equation System

Mass Fraction Fluid Robin Coupled Flux For Mass_Balance [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshEx* | Opposing *OpposingMeshExtent*]= Mass_Fraction_Fluid_Robin_Coupled [Using Data Specification *Data Spec Name*][Mass_Fraction_Field = *mass_fraction_field* | Mass_Fraction_Diffusive_Flux_Field = *mass_fraction_diffusive_flux_field* | Coeff_Field = *coeff_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mass_fraction_field	"string"	undefined
mass_fraction_diffusive_flux_field	"string"	undefined
coeff_field	"string"	undefined

Summary Boundary condition for loose coupling of scalars to a free fluid region

9.2.74 Pyrolysis

Scope: Equation System

```
Pyrolysis Disting For Solid_Z [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Pyrolysis [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for SOLID_z distinguishing condition

9.2.75 Recession

Scope: Equation System

```
Recession Disting For Solid_Z [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Recession [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for SOLID_x distinguishing condition

9.2.76 Kinematic

Scope: Equation System

Kinematic Disting For Solid_Z [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for SOLID_z distinguishing condition

9.2.77 Simple Interp Vector

Scope: Equation System

Simple Interp Vector Flux For Cvfem_Lumped_Div_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Simple_Interp_Vector [Using Data Specification *Data Spec Name*][Source = *source*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
source	"string"	undefined

Summary CVFEM lumped divergence projection simple interpolation vector flux BC

9.2.78 Simple Interp

Scope: Equation System

Simple Interp Flux For Cvfem_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM projection simple interpolation flux BC

9.2.79 Kinematic

Scope: Equation System

Kinematic Rotated For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for rotated momentum

9.2.80 Spring Force

Scope: Equation System

Spring Force Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Spring_Force [Using Data Specification *Data Spec Name*][Mult = *Mult*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Mult	real	undefined

Summary Spring force for momentum boundary condition

9.2.81 Simple Spring Force

Scope: Equation System

Simple Spring Force Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Spring_Force [Using Data Specification *Data Spec Name*][Kx = *kx* |Ky = *ky* |Kz = *kz* |X0 = *x0* |Y0 = *y0* |Z0 = *z0*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
kx	real	undefined
ky	real	undefined
kz	real	undefined
x0	real	undefined
y0	real	undefined
z0	real	undefined

Summary Spring force for momentum boundary condition

9.2.82 Slip Length

Scope: Equation System

Slip Length Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Slip_Length [Using Data Specification *Data Spec Name*][Vs_X = *vs_x* |Vs_Y = *vs_y* |Vs_Z = *vs_z* |Beta = *beta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
vs_x	real	undefined
vs_y	real	undefined
vs_z	real	undefined
beta	real	undefined

Summary Slip length momentum flux

9.2.83 Interface

Scope: Equation System

Interface Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Interface [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary INTERFACE CVFEM momentum flux

9.2.84 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure* |Coflow_Entrainment_Value_X = *coflow_entrainment_value_x* |Coflow_Entrainment_Value_Y = *coflow_entrainment_value_y* |Coflow_Entrainment_Value_Z = *coflow_entrainm*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
coflow_entrainment_value_x	real	undefined
coflow_entrainment_value_y	real	undefined
coflow_entrainment_value_z	real	undefined

Summary CVFEM momentum open advective flow flux BC

9.2.85 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Total_Mdot = *total_mdot* |Temperature = *temperature* |Pressure = *pressure* |Mass_Fraction_0 = *mass_fraction_0* |Mass_Fraction_1 = *mass_fraction_1* |Mass_Fraction_2 = *mass_fraction_2* |Mass_Fraction_3 = *mass_fraction_3* |Mass_Fraction_4 = *mass_fraction_4* |Mass_Fraction_5 = *mass_fraction_5* |Mass_Fraction_ = *mass_fraction_6* |Mass_Fraction_7 = *mass_fraction_7* |Mass_Fraction_8 = *mass_fraction_8*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_mdot	real	undefined
temperature	real	undefined
pressure	real	undefined
mass_fraction_0	real	undefined
mass_fraction_1	real	undefined
mass_fraction_2	real	undefined
mass_fraction_3	real	undefined
mass_fraction_4	real	undefined
mass_fraction_5	real	undefined
mass_fraction_6	real	undefined
mass_fraction_7	real	undefined
mass_fraction_8	real	undefined

9.2.86 Wall Function

Scope: Equation System

Wall Function Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing

OpposingMeshExtent]= Wall_Function [Using Data Specification *Data Spec Name*][Wall_Friction_Factor = *wall_friction_factor* |Wall_Velocity_X = *wall_velocity_x* |Wall_Velocity_Y = *wall_velocity_y* |Wall_Velocity_Z = *wall_velocity_z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
wall_friction_factor	real	undefined
wall_velocity_x	real	undefined
wall_velocity_y	real	undefined
wall_velocity_z	real	undefined

Summary CVFEM momentum wall function flux BC

9.2.87 Symmetry Flow

Scope: Equation System

Symmetry Flow Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Symmetry_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM momentum symmetry flow flux BC

9.2.88 Kinematic

Scope: Equation System

Kinematic Rotated For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for rotated cvfem momentum

9.2.89 Slip

Scope: Equation System

Slip Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Slip [Using Data Specification *Data Spec Name*][Vs_X = *vs_x* | Vs_Y = *vs_y* | Vs_Z = *vs_z* | Beta = *beta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
vs_x	real	undefined
vs_y	real	undefined
vs_z	real	undefined
beta	real	undefined

Summary Slip momentum flux

9.2.90 Ls Capillary

Scope: Equation System

Ls Capillary Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Ls_Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary LS capillary momentum flux

9.2.91 Darcy Slip

Scope: Equation System

Darcy Slip Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Darcy_Slip [Using Data Specification *Data Spec Name*][Vs_X = *vs_x* | Vs_Y = *vs_y* | Vs_Z = *vs_z* | K = *k*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
vs_x	real	undefined
vs_y	real	undefined
vs_z	real	undefined
k	real	undefined

Summary Darcy slip momentum flux

9.2.92 Oriented Slip

Scope: Equation System

Oriented Slip Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Oriented_Slip [Using Data Specification *Data Spec Name*][Vs_X = *vs_x* | Vs_Y = *vs_y* | Vs_Z = *vs_z* | Beta_Normal = *beta_normal* | Beta_Tangent = *beta_tangent*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
vs_x	real	undefined
vs_y	real	undefined
vs_z	real	undefined
beta_normal	real	undefined
beta_tangent	real	undefined

Summary Oriented slip momentum flux

9.2.93 Ls Oriented Slip

Scope: Equation System

Ls Oriented Slip Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing

OpposingMeshExtent]= Ls_Oriented_Slip [Using Data Specification *Data Spec Name*][Vs_X = *vs_x* |Vs_Y = *vs_y* |Vs_Z = *vs_z* |Beta_Normal_A = *beta_normal_a* |Beta_Tangent_A = *beta_tangent_a* |Beta_Normal_B = *beta_normal_b* |Beta_Tangent_B = *beta_tangent_b*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
vs_x	real	undefined
vs_y	real	undefined
vs_z	real	undefined
beta_normal_a	real	undefined
beta_tangent_a	real	undefined
beta_normal_b	real	undefined
beta_tangent_b	real	undefined

Summary Oriented slip momentum flux

9.2.94 Slip Length

Scope: Equation System

Slip Length Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Slip_Length [Using Data Specification *Data Spec Name*][Vs_X = *vs_x* |Vs_Y = *vs_y* |Vs_Z = *vs_z* |Beta = *beta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
vs_x	real	undefined
vs_y	real	undefined
vs_z	real	undefined
beta	real	undefined

Summary Slip length momentum flux

9.2.95 Lens Particles Pressure

Scope: Equation System

Lens Particles Pressure Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Lens_Particles_Pressure [Using Data Specification *Data Spec Name*][Src_X = *src_x* |Dir_X = *dir_x* |Vel_X = *vel_x* |Src_Y = *src_y* |Dir_Y = *dir_y* |Vel_Y = *vel_y*

|Src_Z = *src_z* |Dir_Z = *dir_z* |Vel_Z = *vel_z* |Massflowrate = *massFlowRate* |Particlespeed = *particleSpeed* |Radius = *radius* |T_Liq = *T_liq* |T_Solid = *T_solid* |T_On = *t_on* |T_Off = *t_off*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
src_x	real	undefined
dir_x	real	undefined
vel_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
vel_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
vel_z	real	undefined
massFlowRate	real	undefined
particleSpeed	real	undefined
radius	real	undefined
T_liq	real	undefined
T_solid	real	undefined
t_on	real	undefined
t_off	real	undefined

Summary LENS particles pressure momentum flux

9.2.96 Flow Hydrostatic

Scope: Equation System

Flow Hydrostatic Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Flow_Hydrostatic [Using Data Specification *Data Spec Name*][P_Ref = *p_ref* |Gx = *gx* |Gy = *gy* |Gz = *gz*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
p_ref	real	undefined
gx	real	undefined
gy	real	undefined
gz	real	undefined

Summary Flow hydrostatic momentum flux

9.2.97 Projected Capillary

Scope: Equation System

Projected Capillary Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Projected_Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Projected capillary momentum flux

9.2.98 Electric Traction

Scope: Equation System

Electric Traction Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Electric_Traction [Using Data Specification *Data Spec Name*][Sign = *sign*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
sign	real	undefined

Summary Electric traction momentum flux

9.2.99 Simple Interp

Scope: Equation System

Simple Interp Flux For Cvfem_Temperature_Edge_Gradient_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM edge-based projection simple interpolation flux BC

9.2.100 Inflow Outflow

Scope: Equation System

Inflow Outflow Flux For Level_Set [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Inflow_Outflow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary level set inflow/outflow condition

9.2.101 Spring Force

Scope: Equation System

Spring Force Flux For Solid [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Spring_Force [Using Data Specification *Data Spec Name*][Mult = *Mult*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Mult	real	undefined

Summary Spring force solid flux

9.2.102 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Specific_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Total_Mdot = *total_mdot* |Inflow_Phi = *inflow_phi*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_mdot	real	undefined
inflow_phi	real	undefined

9.2.103 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Cvfem_Specific_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure* |Far_Field_Entrainment_Value = *far_field_entrainm*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

9.2.104 Porous Robin One Region

Scope: Equation System

Porous Robin One Region Flux For Cvfem_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Porous_Robin_One_Region [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Boundary condition for loose coupling of continuity to a free fluid region

9.2.105 Interface

Scope: Equation System

Interface Flux For Cvfem_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Interface [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary INTERFACE CVFEM continuity flux

9.2.106 Wall Function

Scope: Equation System

Wall Function Disting For Hfem_Specific_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Wall_Function [Using Data Specification *Data Spec Name*][Scale_Factor = *scale_factor* |Wall_Friction_Factor = *Wall_Friction_Factor* |Wall_Velocity_X = *Wall_Velocity_X* |Wall_Velocity_Y = *Wall_Velocity_Y* |Wall_Velocity_Z = *Wall_Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
scale_factor	real	undefined
Wall_Friction_Factor	real	undefined
Wall_Velocity_X	real	undefined
Wall_Velocity_Y	real	undefined
Wall_Velocity_Z	real	undefined

Summary HFEM specific dissipation rate distinguishing condition

9.2.107 Low Reynolds

Scope: Equation System

Low Reynolds Disting For Hfem_Specific_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx*

|Opposing *OpposingMeshExtent*]= Low_Reynolds [Using Data Specification *Data Spec Name*][Scale_Factor = *scale_factor*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
scale_factor	real	undefined

Summary HFEM specific dissipation rate distinguishing condition

9.2.108 Gaussian Spot Power Weld

Scope: Equation System

Gaussian Spot Power Weld Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Gaussian_Spot_Power_Weld [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Src_X = *src_x* | Dir_X = *dir_x* |Src_Y = *src_y* |Dir_Y = *dir_y* |Src_Z = *src_z* |Dir_Z = *dir_z* |R = *r* |T_On = *t_on* |T_Off = *t_off* |R_Eff = *r_eff* |Alpha = *alpha* |Depth_Enhanced_Absorption_Data = *depth_enhanced_* |Compute_Visibility_Field = *compute_visibility_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
src_x	real	undefined
dir_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
r	real	undefined
t_on	real	undefined
t_off	real	undefined
r_eff	real	undefined
alpha	real	undefined
depth_enhanced_absorption_data	"string"	undefined
compute_visibility_field	"string"	undefined

9.2.109 Gaussian Line Power Weld

Scope: Equation System

Gaussian Line Power Weld Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Gaussian_Line_Power_Weld [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Src_X = *src_x* |Dir_X = *dir_x* |Vel_X = *vel_x* |Src_Y = *src_y* |Dir_Y = *dir_y* |Vel_Y = *vel_y* |Src_Z = *src_z* |Dir_Z = *dir_z* |Vel_Z = *vel_z* |R = *r* |T_On = *t_on* |T_Off = *t_off* |R_Eff = *r_eff* |Alpha = *alpha* |Depth_Enhanced_Absorption_Data = *depth_enhanced_absorption_data* |Compute_Visibility_Field = *compute_visibility_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
src_x	real	undefined
dir_x	real	undefined
vel_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
vel_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
vel_z	real	undefined
r	real	undefined
t_on	real	undefined
t_off	real	undefined
r_eff	real	undefined
alpha	real	undefined
depth_enhanced_absorption_data	"string"	undefined
compute_visibility_field	"string"	undefined

9.2.110 Sharp Spot Weld

Scope: Equation System

Sharp Spot Weld Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Sharp_Spot_Weld [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Src_X = *src_x* |Dir_X = *dir_x* |Src_Y = *src_y* |Dir_Y = *dir_y* |Src_Z = *src_z* |Dir_Z = *dir_z* |R = *r* |T_On = *t_on* |T_Off = *t_off* |Flux = *flux*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
src_x	real	undefined
dir_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
r	real	undefined
t_on	real	undefined
t_off	real	undefined
flux	real	undefined

9.2.111 Surface Stabilization

Scope: Equation System

```
Surface Stabilization Fluxbp For Energy [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Surface_Stabilization [ Using Data Specification Data Spec Name ][ D
= d ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
d	real	undefined

Summary surface diffusion term for energy

9.2.112 Normal Capillary Stabilization

Scope: Equation System

```
Normal Capillary Stabilization Fluxbp For Momentum [ {of|species|subindex} Species |{in
|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent
|Opposing OpposingMeshExtent ]= Normal_Capillary_Stabilization [ Using Data Specification
Data Spec Name ][ Mu0 = mu0 |Multiplier = multiplier |Use_Velocity_Correction = use_velocity_correctio
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC

9.2.113 Pid Controlled Cooler

Scope: Equation System

Pid Controlled Cooler Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pid_Controlled_Cooler [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |P = *p* |I = *i* |D = *d* |Band = *Band* |Filter_Tau = *Filter_Tau* |Setpoint_Function_Name = *Setpoint_Function_Name* |Control_Variable = *Control_Variable* |Max_Cooling_Power = *Max_Cooling_Power*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
p	real	undefined
i	real	undefined
d	real	undefined
Band	real	undefined
Filter_Tau	real	undefined
Setpoint_Function_Name	"string"	undefined
Control_Variable	"string"	undefined
Max_Cooling_Power	real	undefined

9.2.114 Pid Controlled Bidirectional

Scope: Equation System

Pid Controlled Bidirectional Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pid_Controlled_Bidirectional [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |P = *p* |I = *i* |D = *d* |Band = *Band* |Filter_Tau = *Filter_Tau* |Setpoint_Function_Name = *Setpoint_Function_Name*

|Control_Variable = *Control_Variable* |Max_Cooling_Power = *Max_Cooling_Power* |Max_Heating_Power = *Max_Heating_Power*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
p	real	undefined
i	real	undefined
d	real	undefined
Band	real	undefined
Filter_Tau	real	undefined
Setpoint_Function_Name	"string"	undefined
Control_Variable	"string"	undefined
Max_Cooling_Power	real	undefined
Max_Heating_Power	real	undefined

9.2.115 Ls Capillary

Scope: Equation System

Ls Capillary Fluxbp For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Ls_Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC using level set normal

9.2.116 Porous Robin Coupled One Region

Scope: Equation System

Porous Robin Coupled One Region Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExt* |Opposing *OpposingMeshExtent*]= Porous_Robin_Coupled_One_Region [Using Data Specification *Data Spec Name*][Data = *data*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined

Summary Boundary condition for loose coupling of enthalpy to a porous region

9.2.117 Symmetry Flow

Scope: Equation System

Symmetry Flow Flux For Cvfem_Solvent_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Symmetry_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM solvent momentum symmetry flow flux BC

9.2.118 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Cvfem_Solvent_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* | Total_Pressure = *total_pressure* | Coflow_Entrainment_Value_X = *coflow_entrainment_value_x* | Coflow_Entrainment_Value_Y = *coflow_entrainment_value_y* | Coflow_Entrainment_Value_Z = *coflow_entrainm*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
coflow_entrainment_value_x	real	undefined
coflow_entrainment_value_y	real	undefined
coflow_entrainment_value_z	real	undefined

Summary CVFEM solvent momentum open advective flow flux BC

9.2.119 Open Flow Convection

Scope: Equation System

Open Flow Convection Flux For Mixture_Fraction [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow_Convection [Using Data Specification *Data Spec Name*][Reference_Value = *Reference_Value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Reference_ Value	"string"	undefined

Summary Open flow convection BC for the mixture fraction equation

9.2.120 Kinematic

Scope: Equation System

Kinematic Rotated For Momentum_Colloc [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_ normal	"string"	undefined
no_ mesh_ movement	"string"	undefined

Summary Kinematic model for rotated collocated momentum boundary condition

9.2.121 Extrapolated Pressure

Scope: Equation System

Extrapolated Pressure Flux For Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Extrapolated_Pressure [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Exrapolated pressure for use with artificial compressibility methods

9.2.122 Pspg

Scope: Equation System

```
Pspg Flux For Continuity [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Pspg [ Using Data Specification Data Spec Name ][ Tausurfacefactor = TauSurfaceFactor |
Tausurfacejacobianfactor = TauSurfaceJacobianFactor ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
TauSurfaceFactor	real	undefined
TauSurfaceJacobianFactor	real	undefined

Summary Continuity PSPG flux BC

9.2.123 Circle Weld

Scope: Equation System

```
Circle Weld Flux For Energy [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Circle_Weld [ Using Data Specification Data Spec Name ][ Power_Output = power_output |
Flux_Output = flux_output |Toggle = toggle |O_X = o_x |O_Y = o_y |O_Z = o_z |Ro_X = ro_x
|Ro_Y = ro_y |Ro_Z = ro_z |A_X = a_x |A_Y = a_y |A_Z = a_z |W = w |Flux = flux |R = r
|Max_Degrees = max_degrees |T_On = t_on |T_Off = t_off |Normal_Tolerance = normal_tolerance
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
o_x	real	undefined
o_y	real	undefined
o_z	real	undefined
ro_x	real	undefined
ro_y	real	undefined
ro_z	real	undefined
a_x	real	undefined
a_y	real	undefined
a_z	real	undefined
w	real	undefined
flux	real	undefined
r	real	undefined
max_degrees	real	undefined
t_on	real	undefined
t_off	real	undefined
normal_tolerance	real	undefined

9.2.124 Distribution Factor

Scope: Equation System

Distribution Factor Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Distribution_Factor [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Attribute_Name = *attribute_name* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
attribute_name	"string"	undefined
multiplier	real	undefined

9.2.125 Recession

Scope: Equation System

Recession Disting For Mesh_Y [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Recession [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* |K = *k* |K_0 = *k_0* |N = *n* |Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for mesh_x distinguishing condition

9.2.126 Pyrolysis

Scope: Equation System

Pyrolysis Disting For Mesh_Y [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pyrolysis [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* |K = *k* |K_0 = *k_0* |N = *n* |Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for mesh_x distinguishing condition

9.2.127 From Cht Temperature

Scope: Equation System

From Cht Temperature Disting For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= From_Cht_Temperature [Using Data Specification *Data Spec Name*][Interface_Name = *Interface_Name* |Twall_Field = *Twall_Field* |Tref_Field = *Tref_Field* |H_Field = *H_Field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Interface_Name	"string"	undefined
Twall_Field	"string"	undefined
Tref_Field	"string"	undefined
H_Field	"string"	undefined

Summary Energy distinguishing condition from conjugate heat transfer temperature

9.2.128 Calore User Sub

Scope: Equation System

```
Calore User Sub Flux For Energy [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Calore_User_Sub [ Using Data Specification Data Spec Name ][ Power_Output = power_output
|Flux_Output = flux_output |Toggle = toggle |Name = name |Type = type |Multiplier = multiplier
|Material_Data_Block = material_data_block |Data = data |Scaling_Field = scaling_field ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

9.2.129 Recession

Scope: Equation System

```
Recession Disting For Level_Set [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Recession [ Using Data Specification Data Spec Name ][ P_Ref = p_ref |K = k |K_0 = k_0
|N = n |Pressure = pressure ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
p_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
pressure	"string"	undefined

Summary Advection model for extension speed distinguishing condition

9.2.130 Calore User Sub

Scope: Equation System

Calore User Sub Flux For Current [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Calore_User_Sub [Using Data Specification *Data Spec Name*][Toggle = *toggle* |Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_blo* |Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
toggle	"string"	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

9.2.131 Constant With Cutoff Voltage

Scope: Equation System

Constant With Cutoff Voltage Flux For Current [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Constant_With_Cutoff_Voltage [Using Data Specification *Data Spec Name*][Toggle = *toggle* |Value = *value* |Cutoff_Value = *cutoff_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
toggle	"string"	undefined
value	real	undefined
cutoff_value	real	undefined

Summary Apply a constant current until a cutoff voltage is crossed anywhere on the surface. After that apply 0 current.

9.2.132 Resistive Load

Scope: Equation System

```
Resistive Load Flux For Current [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Resistive_Load [ Using Data Specification Data Spec Name ][ Toggle = toggle |Area = Area
|Referencevoltage = ReferenceVoltage ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
toggle	"string"	undefined
Area	real	undefined
ReferenceVoltage	real	undefined

Summary Model a boundary attached to a resistive load. The local applied current density is V / (R A) where V is the local voltage, R is the specified resistance, and A is the area of the surface the BC is applied to.

9.2.133 Simple Interp

Scope: Equation System

```
Simple Interp Flux For Cvfem_Lumped_Projection [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Simple_Interp [ Using Data Specification Data Spec Name ][ ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM lumped projection simple interpolation flux BC

9.2.134 Open

Scope: Equation System

Open Flux For Cvfem_Lumped_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open [Using Data Specification *Data Spec Name*][Total_Pressure = *total_pressure* |Pressure = *pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_pressure	real	undefined
pressure	real	undefined

Summary CVFEM lumped projection open flux BC

9.2.135 Free Open Flow

Scope: Equation System

Free Open Flow Flux For Cvfem_Turbulent_Kinetic_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Free_Open_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.136 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Turbulent_Kinetic_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Total_Mdot = *total_mdot* |Inflow_Phi = *inflow_phi*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_mdot	real	undefined
inflow_phi	real	undefined

9.2.137 Open Adv Flow

Scope: Equation System

```
Open Adv Flow Flux For Cvfem_Turbulent_Kinetic_Energy [ {of|species|subindex} Species |
{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExt
|Opposing OpposingMeshExtent ]= Open_Adv_Flow [ Using Data Specification Data Spec Name ][
Pressure = pressure |Total_Pressure = total_pressure |Far_Field_Entrainment_Value = far_field_entrainm
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

9.2.138 User Vector Field Influx

Scope: Equation System

```
User Vector Field Influx Flux For Energy [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= User_Vector_Field_Influx [ Using Data Specification Data Spec Name ][
Power_Output = power_output |Flux_Output = flux_output |Toggle = toggle |Name = name |Multiplier
= multiplier |Scaling = scaling |Global_Var = global_var |Time_Function = time_function ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
name	"string"	undefined
multiplier	real	undefined
scaling	real	undefined
global_var	"string"	undefined
time_function	"string"	undefined

Summary User vector field influx energy flux

9.2.139 Scaled Nat Conv

Scope: Equation System

Scaled Nat Conv Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Scaled_Nat_Conv [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Multiplier = *multiplier* |Scaling_Field = *scaling_field* |Scaling = *scaling* |Global_Var = *global_var* |Time_Function = *time_function*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined
scaling_field	"string"	undefined
scaling	real	undefined
global_var	"string"	undefined
time_function	"string"	undefined

Summary Scaled natural convection energy flux

9.2.140 Nat Conv

Scope: Equation System

Nat Conv Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*

]= Nat_Conv [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |H = *h* |T_Ref = *t_ref* |Advective_Bar = *advective_bar* |Command_Block_N = *command_block_name*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
h	"string"	undefined
t_ref	"string"	undefined
advective_bar	"string"	undefined
command_block_name	"string"	undefined

Summary Convective energy flux defined with constant models for heat transfer coefficient and reference temperature

9.2.141 Latent Heat

Scope: Equation System

Latent Heat Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Latent_Heat [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Yinf = *Yinf*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
Yinf	real	undefined

Summary Latent heat energy flux

9.2.142 Generalized Rad

Scope: Equation System

Generalized Rad Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Generalized_Rad [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined

Summary Radiative energy flux, possibly defined with different models for emissivity, form factor and irradiation

9.2.143 Kuntz

Scope: Equation System

Kuntz Flux For Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Kuntz [Using Data Specification *Data Spec Name*][Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Multiplier = *multiplier* | File = *file* | Units = *units*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined
file	"string"	undefined
units	"string"	undefined

Summary Interpolated value from tabulated data for the energy flux

9.2.144 Exponential Vapor Cooling

Scope: Equation System

Exponential Vapor Cooling Flux For Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Exponential_Vapor_Cooling [Using Data Specification *Data Spec Name*][Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Tboil = *tboil* | L = *l* | Pa = *pa* | Mw = *mw* | Alpha = *alpha*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
tboil	real	undefined
l	real	undefined
pa	real	undefined
mw	real	undefined
alpha	real	undefined

Summary Exponential vapor cooling energy flux

9.2.145 User Vector Field

Scope: Equation System

User Vector Field Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= User_Vector_Field [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Name = *name* |Multiplier = *multiplier* |Scaling = *scaling* |Global_Var = *global_var* |Time_Function = *time_function*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
name	"string"	undefined
multiplier	real	undefined
scaling	real	undefined
global_var	"string"	undefined
time_function	"string"	undefined

Summary User vector field energy flux

9.2.146 Thermal Latent Heat

Scope: Equation System

Thermal Latent Heat Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing

OpposingMeshExtent]= Thermal_Latent_Heat [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined

Summary Thermal latent heat energy flux

9.2.147 Vapor Cooling

Scope: Equation System

Vapor Cooling Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Vapor_Cooling [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Tboil = *tboil*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
tboil	real	undefined

Summary Vapor cooling energy flux

9.2.148 Generalized Rad

Scope: Equation System

Generalized Rad Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Generalized_Rad [Using Data Specification *Data Spec Name*][Data = *data* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
multiplier	real	undefined

Summary Radiative energy flux, possibly defined with different models for emissivity, form factor and irradiation

9.2.149 Nat Conv

Scope: Equation System

```
   Nat Conv Flux For Cvfem_Energy [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Nat_Conv [ Using Data Specification Data Spec Name ][ Data = data |H = h |T_Ref = t_ref
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
h	"string"	undefined
t_ref	"string"	undefined

Summary CVFEM convective energy flux defined with constant models for heat transfer coefficient and reference temperature

9.2.150 Open Adv Flow

Scope: Equation System

```
   Open Adv Flow Flux For Cvfem_Energy [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Open_Adv_Flow [ Using Data Specification Data Spec Name ][ Data = data
|Far_Field_Entrainment_Value = Far_Field_Entrainment_Value |Pressure = pressure |Total_Pressure
= total_pressure ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
Far_Field_Entrainment_Value	real	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary CVFEM energy open advective flow flux

9.2.151 Open Adv Flow From Temperature

Scope: Equation System

Open Adv Flow From Temperature Flux For Cvfem_Energy [{of|species|subindex} *Species* | {in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExt* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow_From_Temperature [Using Data Specification *Data Spec Name*][Data = *data* |Temperature = *temperature* |Pressure = *pressure* |Mass_Fraction_0 = *mass_fraction_0* |Mass_Fraction_1 = *mass_fraction_1* |Mass_Fraction_2 = *mass_fraction_2* | Mass_Fraction_3 = *mass_fraction_3* |Mass_Fraction_4 = *mass_fraction_4* |Mass_Fraction_5 = *mass_fraction_* |Mass_Fraction_6 = *mass_fraction_6* |Mass_Fraction_7 = *mass_fraction_7* |Mass_Fraction_8 = *mass_fraction_8* |Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
temperature	real	undefined
pressure	real	undefined
mass_fraction_0	real	undefined
mass_fraction_1	real	undefined
mass_fraction_2	real	undefined
mass_fraction_3	real	undefined
mass_fraction_4	real	undefined
mass_fraction_5	real	undefined
mass_fraction_6	real	undefined
mass_fraction_7	real	undefined
mass_fraction_8	real	undefined
total_pressure	real	undefined

Summary CVFEM energy open advective flow from temperature flux

9.2.152 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Data = *data* |Inflow_Phi = *Inflow_Phi* |Total_Mdot = *total_mdot*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
Inflow_Phi	real	undefined
total_mdot	real	undefined

Summary CVFEM energy mass flux flux BC

9.2.153 Mass Flux From Temperature

Scope: Equation System

Mass Flux From Temperature Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux_From_Temperature [Using Data Specification *Data Spec Name*][Data = *data* |Temperature = *temperature* |Pressure = *pressure* |Mass_Fraction_0 = *mass_fraction_0* |Mass_Fraction_1 = *mass_fraction_1* |Mass_Fraction_2 = *mass_fraction_2* |Mass_Fraction_3 = *mass_fraction_3* |Mass_Fraction_4 = *mass_fraction_4* |Mass_Fraction_5 = *mass_fraction_* |Mass_Fraction_6 = *mass_fraction_6* |Mass_Fraction_7 = *mass_fraction_7* |Mass_Fraction_8 = *mass_fraction_8* |Total_Mdot = *total_mdot*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
temperature	real	undefined
pressure	real	undefined
mass_fraction_0	real	undefined
mass_fraction_1	real	undefined
mass_fraction_2	real	undefined
mass_fraction_3	real	undefined
mass_fraction_4	real	undefined
mass_fraction_5	real	undefined
mass_fraction_6	real	undefined
mass_fraction_7	real	undefined
mass_fraction_8	real	undefined
total_mdot	real	undefined

Summary CVFEM energy mass flux from temperature flux BC

9.2.154 Generalized Nat Conv

Scope: Equation System

Generalized Nat Conv Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Generalized_Nat_Conv [Using Data Specification *Data Spec Name*][Data = *data* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
multiplier	real	undefined

Summary CVFEM energy generalized natural convection flux BC

9.2.155 Nat Conv

Scope: Equation System

Nat Conv Flux For Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Nat_Conv [Using Data Specification *Data Spec Name*][H = *h* |T_Ref = *t_ref*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
h	real	undefined
t_ref	real	undefined

Summary Natural convection BC for enthalpy flux

9.2.156 Latent Heat

Scope: Equation System

Latent Heat Flux For Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Latent_Heat [Using Data Specification *Data Spec Name*][Yinf = *Yinf*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Yinf	real	undefined

Summary Latent heat enthalpy flux

9.2.157 Porous Robin Coupled With Solid Phase Convection

Scope: Equation System

Porous Robin Coupled With Solid Phase Convection Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Porous_Robin_Coupled_With_Solid_Phase_Convection [Using Data Specification *Data Spec Name*][Data = *data* |Multiplier = *multiplier* |Enthalpy_Field = *enthalpy_field* |Enthalpy_Diffusive_Flux_Field = *enthalpy_diffusive_flux_field* |Solid_Phase_Temperatu = *solid_phase_temperature_field* |Porosity_Field = *porosity_field* |Averaged_Coeff_Field = *averaged_coef* |Volumetric_Heat_Transfer_Coeff_Field = *volumetric_heat_transfer_coeff_field* |Specific_Surface_Area_Fie = *specific_surface_area_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
multiplier	real	undefined
enthalpy_field	"string"	undefined
enthalpy_diffusive_flux_field	"string"	undefined
solid_phase_temperature_field	"string"	undefined
porosity_field	"string"	undefined
averaged_coeff_field	"string"	undefined
volumetric_heat_transfer_coeff_field	"string"	undefined
specific_surface_area_field	"string"	undefined

Summary Boundary condition for loose coupling of enthalpy to a porous region

9.2.158 Interface

Scope: Equation System

Interface Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Interface [Using Data Specification *Data Spec Name*][Data = *data*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined

Summary INTERFACE CVFEM energy flux

9.2.159 Porous Robin Coupled With Solid Phase Convection One Region

Scope: Equation System

Porous Robin Coupled With Solid Phase Convection One Region Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Porous_Robin_Coupled_With_So [Using Data Specification *Data Spec Name*][Data = *data* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
multiplier	real	undefined

Summary Boundary condition for coupling of enthalpy to a porous region in the same region.

9.2.160 Nat E

Scope: Equation System

Nat E Flux For Charge_Density [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Nat_E [Using Data Specification *Data Spec Name*][Sigma = *sigma*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
sigma	"string"	undefined

Summary Natural J = conductivity times Electric field BC that falls out of flux diffusion term

9.2.161 Laser Weld

Scope: Equation System

Laser Weld Flux For Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Laser_Weld [Using Data Specification *Data Spec Name*][Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Flux = *flux* | R = *r* | Normal_Tolerance = *normal_tolerance* | Path_Function = *path_function*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
flux	real	undefined
r	real	undefined
normal_tolerance	real	undefined
path_function	"string"	undefined

9.2.162 Rad

Scope: Equation System

Rad Flux For Cvfem_Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Rad [Using Data Specification *Data Spec Name*][Data = *data* | T_Ref = *t_ref* | Crad = *crad*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
t_ref	real	undefined
crad	real	undefined

Summary CVFEM radiative energy flux defined with constant models for emissivity, form factor and reference temperature

9.2.163 From Cht Temperature Cpt

Scope: Equation System

From Cht Temperature Cpt Disting For Cvfem_Energy [{of | species | subindex} *Species* | {in

|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= From_Cht_Temperature_Cpt [Using Data Specification *Data Spec Name*][Twall_Field = *Twall_Field* |Variable = *variable* |Order = *order* |Variable_Offset = *variable_offset* |C0 = *c0* |C1 = *c1* |C2 = *c2* |C3 = *c3* |C4 = *c4* |C5 = *c5* |C6 = *c6* |C7 = *c7* |C8 = *c8*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Twall_Field	"string"	undefined
variable	"string"	undefined
order	integer	undefined
variable_offset	real	undefined
c0	real	undefined
c1	real	undefined
c2	real	undefined
c3	real	undefined
c4	real	undefined
c5	real	undefined
c6	real	undefined
c7	real	undefined
c8	real	undefined

Summary CVFEM energy distinguishing condition from CHT temperature using Cp*T model

9.2.164 From Cht Temperature

Scope: Equation System

From Cht Temperature Disting For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= From_Cht_Temperature [Using Data Specification *Data Spec Name*][Twall_Field = *Twall_Field* |Variable = *variable* |Order = *order* |Variable_Offset = *variable_offset* |C0 = *c0* |C1 = *c1* |C2 = *c2* |C3 = *c3* |C4 = *c4* |C5 = *c5* |C6 = *c6* |C7 = *c7* |C8 = *c8* |Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Twall_Field	"string"	undefined
variable	"string"	undefined
order	integer	undefined
variable_offset	real	undefined
c0	real	undefined
c1	real	undefined
c2	real	undefined
c3	real	undefined
c4	real	undefined
c5	real	undefined
c6	real	undefined
c7	real	undefined
c8	real	undefined
poffset	real	undefined

Summary CVFEM energy distinguishing condition from CHT temperature using Cantera

9.2.165 Lens Deposition

Scope: Equation System

```
Lens Deposition Disting For Extension_Speed [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Lens_Deposition [ Using Data Specification Data Spec Name ][ Src_X =
src_x |Dir_X = dir_x |Vel_X = vel_x |Src_Y = src_y |Dir_Y = dir_y |Vel_Y = vel_y |Src_Z
= src_z |Dir_Z = dir_z |Vel_Z = vel_z |Massflowrate = massFlowRate |Particledensity = particleDensit
|Radius = radius |T_Liq = T_liq |T_Solid = T_solid |T_On = t_on |T_Off = t_off ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
src_x	real	undefined
dir_x	real	undefined
vel_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
vel_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
vel_z	real	undefined
massFlowRate	real	undefined
particleDensity	real	undefined
radius	real	undefined
T_liq	real	undefined
T_solid	real	undefined
t_on	real	undefined
t_off	real	undefined

Summary LENS Deposition mass source model for extension speed distinguishing condition

9.2.166 Pyrolysis

Scope: Equation System

Pyrolysis Disting For Extension_Speed [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pyrolysis [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* |K = *k* |K_0 = *k_0* |N = *n* |Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for extension speed distinguishing condition

9.2.167 Solidification

Scope: Equation System

Solidification Disting For Extension_Speed [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Solidification [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Solidification model for extension speed distinguishing condition

9.2.168 Advection

Scope: Equation System

Advection Disting For Extension_Speed [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Advection [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Advection model for extension speed distinguishing condition

9.2.169 Recession

Scope: Equation System

Recession Disting For Extension_Speed [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Recession [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* |K = *k* |K_0 = *k_0* |N = *n* |Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Advection model for extension speed distinguishing condition

9.2.170 Melting

Scope: Equation System

Melting Disting For Extension_Speed [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Melting [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Melting model for extension speed distinguishing condition

9.2.171 Simple Interp

Scope: Equation System

Simple Interp Flux For Hfem_Bf_Lumped_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Simple interp model for BF projection flux

9.2.172 Suspension Traction

Scope: Equation System

Suspension Traction Flux For Div_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Suspension_Traction [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Suspension traction for suspension balance model

9.2.173 Curvature

Scope: Equation System

Curvature Flux For Div_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Curvature [Using Data Specification *Data Spec Name*][Theta = *theta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
theta	real	undefined

Summary Curvature model for div projection flux

9.2.174 Capillary Force

Scope: Equation System

Capillary Force Flux For Div_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Capillary_Force [Using Data Specification *Data Spec Name*][Theta = *theta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
theta	real	undefined

Summary Capillary force model for div projection flux

9.2.175 Simple Interp

Scope: Equation System

Simple Interp Flux For Darcy_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Darcy momentum simple interpolation flux BC

9.2.176 Kinematic

Scope: Equation System

Kinematic Disting For Mesh_X [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for mesh_x distinguishing condition

9.2.177 Recession

Scope: Equation System

```
  Recession Disting For Mesh_X [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Recession [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for mesh_x distinguishing condition

9.2.178 Pyrolysis

Scope: Equation System

```
  Pyrolysis Disting For Mesh_X [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Pyrolysis [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for mesh_x distinguishing condition

9.2.179 Simple Inflow

Scope: Equation System

Simple Inflow Flux For Cvfem_Dispersed_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Inflow [Using Data Specification *Data Spec Name*][Velocity_X = *Velocity_X* |Velocity_Y = *Velocity_Y* |Velocity_Z = *Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Velocity_X	real	undefined
Velocity_Y	real	undefined
Velocity_Z	real	undefined

Summary Inflow mass flux for dispersed continuity equation; specified density and velocity

9.2.180 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Cvfem_Dispersed_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Coflow_Entrainment_Valuex = *Coflow_Entrainment_ValueX* |Coflow_Entrainment_Valuey = *Coflow_Entrainment_*|Coflow_Entrainment_Valuez = *Coflow_Entrainment_ValueZ*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Coflow_Entrainment_ValueX	real	undefined
Coflow_Entrainment_ValueY	real	undefined
Coflow_Entrainment_ValueZ	real	undefined

9.2.181 Inflow

Scope: Equation System

Inflow Flux For Hfem_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Inflow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary IBP continity inflow; nodal interp

9.2.182 Symmetry Flow

Scope: Equation System

Symmetry Flow Flux For Cvfem_Dispersed_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Symmetry_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.183 Open Flow

Scope: Equation System

Open Flow Flux For Cvfem_Dispersed_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.184 Simple Interp Tensor

Scope: Equation System

Simple Interp Tensor Flux For Cvfem_Div_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp_Tensor [Using Data Specification *Data Spec Name*][Source = *source*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
source	"string"	undefined

Summary CVFEM divergence projection simple interpolation tensor flux BC

9.2.185 Simple Interp Vector

Scope: Equation System

```
Simple Interp Vector Flux For Cvfem_Div_Projection [ {of|species|subindex} Species |{in
|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent
|Opposing OpposingMeshExtent ]= Simple_Interp_Vector [ Using Data Specification Data Spec
Name ][ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
source	"string"	undefined

Summary CVFEM divergence projection simple interpolation vector flux BC

9.2.186 From Cht Temperature

Scope: Equation System

```
From Cht Temperature Disting For Porous_Enthalpy [ {of|species|subindex} Species |{in|
material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent
|Opposing OpposingMeshExtent ]= From_Cht_Temperature [ Using Data Specification Data Spec
Name ][ Interface_Name = Interface_Name |Twall_Field = Twall_Field |Tref_Field = Tref_Field
|H_Field = H_Field ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Interface_Name	"string"	undefined
Twall_Field	"string"	undefined
Tref_Field	"string"	undefined
H_Field	"string"	undefined

Summary Porous enthalpy distinguishing condition from conjugate heat transfer temperature

9.2.187 Capillary

Scope: Equation System

Capillary Fluxbp For Hfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC

9.2.188 Free Open Flow

Scope: Equation System

Free Open Flow Flux For Hfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Free_Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Pressure User Sub *PRESSURE User Sub*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
PRESSURE User Sub	string	undefined

Summary Open flow for momentum; apply the full stress

9.2.189 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Hfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure* |Coflow_Entrainment_Value_X = *coflow_entrainment_value_x* |Coflow_Entrainment_Value_Y = *coflow_entrainment_value_y* |Coflow_Entrainment_Value_Z = *coflow_entrainm*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
coflow_entrainment_value_x	real	undefined
coflow_entrainment_value_y	real	undefined
coflow_entrainment_value_z	real	undefined

Summary HFEM momentum open advective flow flux BC

9.2.190 Outflow

Scope: Equation System

Outflow Disting For Level_Set [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Outflow [Using Data Specification *Data Spec Name*][V0 = *v0*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined

Summary Outflow model for level set distinguishing condition

9.2.191 Generalized Rad

Scope: Equation System

Generalized Rad Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Generalized_Rad [Using Data Specification *Data Spec Name*][Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
multiplier	real	undefined

Summary Radiative porous enthalpy flux, possibly defined with different models for emissivity, form factor and irradiation

9.2.192 Fluid Robin Coupled With Solid Convection One Region

Scope: Equation System

Fluid Robin Coupled With Solid Convection One Region Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Fluid_Robin_Coupled_With_Solid_Conv [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Boundary condition for loose coupling of enthalpy to a free fluid region

9.2.193 Enclosure Radiation

Scope: Equation System

Enclosure Radiation Flux For Porous_Enthalpy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Enclosure_Radiation [Using Data Specification *Data Spec Name*][Ndof = *ndof* |Multiplier = *multiplier* |Enclosure = *enclosure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
ndof	integer	undefined
multiplier	real	undefined
enclosure	"string"	undefined

9.2.194 Curvature

Scope: Equation System

Curvature Flux For Lumped_Div_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Curvature [Using Data Specification *Data Spec Name*][Theta = *theta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
theta	real	undefined

Summary Curvature model for lumped div projection flux

9.2.195 Shear Free

Scope: Equation System

Shear Free Flux For Solid [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Shear_Free [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Shear-free traction solid flux

9.2.196 Capillary

Scope: Equation System

Capillary Fluxbp For Solid [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Solid capillary flux BP BC

9.2.197 Pressure

Scope: Equation System

Pressure Flux For Solid [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pressure [Using Data Specification *Data Spec Name*][User_Field_Name = *user_field_name* |Use_Bulk_Node = *use_bulk_node* |P = *p* |C_T = *c_t* |C_X = *c_x* |C_Y = *c_y* |C_Z = *c_z* |C_Cos_W = *c_cos_w* |C_Sin_W = *c_sin_w* |C1_Cos = *c1_cos* |C1_Sin = *c1_sin*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
user_field_name	"string"	undefined
use_bulk_node	"string"	undefined
p	real	undefined
c_t	real	undefined
c_x	real	undefined
c_y	real	undefined
c_z	real	undefined
c_cos_w	real	undefined
c_sin_w	real	undefined
c1_cos	real	undefined
c1_sin	real	undefined

Summary Pressure solid flux

9.2.198 Pressure User Function

Scope: Equation System

Pressure User Function Flux For Solid [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pressure_User_Function [Using Data Specification *Data Spec Name*][Name = *name* |X = *x* |X_Multiplier = *x_multiplier* |Multiplier = *multiplier* |Toggle = *toggle*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
x	"string"	undefined
x_multiplier	real	undefined
multiplier	real	undefined
toggle	"string"	undefined

Summary Pressure user function solid flux

9.2.199 Constant Traction

Scope: Equation System

Constant Traction Flux For Solid [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Constant_Traction [Using Data Specification *Data Spec Name*][X = *x* |Y = *y* |Z = *z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
x	"string"	undefined
y	"string"	undefined
z	"string"	undefined

Summary Constant traction solid flux

9.2.200 Electric Traction

Scope: Equation System

Electric Traction Flux For Solid [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Electric_Traction [Using Data Specification *Data Spec Name*][Sign = *sign*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
sign	real	undefined

Summary Electric traction solid flux

9.2.201 Wall Function

Scope: Equation System

Wall Function Flux For Hfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Wall_Function [Using Data Specification *Data Spec Name*][Scale_Factor = *scale_factor* |Wall_Friction_Factor = *wall_friction_factor* |Wall_Velocity_X = *wall_velocity_x* |Wall_Velocity_Y = *wall_velocity_y* |Wall_Velocity_Z = *wall_velocity_z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
scale_factor	real	undefined
wall_friction_factor	real	undefined
wall_velocity_x	real	undefined
wall_velocity_y	real	undefined
wall_velocity_z	real	undefined

Summary HFEM momentum symmetry flow flux BC

9.2.202 Open Adv Flow

Scope: Equation System

```
   Open Adv Flow Flux For Hfem_Turbulent_Kinetic_Energy [ {of|species|subindex} Species |
{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExt
|Opposing OpposingMeshExtent ]= Open_Adv_Flow [ Using Data Specification Data Spec Name ][
Pressure = pressure |Total_Pressure = total_pressure |Far_Field_Entrainment_Value = far_field_entrainm
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

Summary HFEM turbulent kinetic energy open advective flow flux BC

9.2.203 Simple Dp Interp

Scope: Equation System

```
   Simple Dp Interp Flux For Hfem_Velocity_Pressure_Projection [ {of|species|subindex} Species
|{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshEx
|Opposing OpposingMeshExtent ]= Simple_Dp_Interp [ Using Data Specification Data Spec Name
][ ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Simple DP interploation model for HFEM velocity pressure projecion flux

9.2.204 Interface

Scope: Equation System

Interface Flux For Hfem_Continuity [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Interface [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary INTERFACE HFEM continuity flux

9.2.205 Simple Inflow

Scope: Equation System

Simple Inflow Flux For Hfem_Continuity [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Simple_Inflow [Using Data Specification *Data Spec Name*][Density = *Density* | Velocity_X = *Velocity_X* | Velocity_Y = *Velocity_Y* | Velocity_Z = *Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Density	real	undefined
Velocity_X	real	undefined
Velocity_Y	real	undefined
Velocity_Z	real	undefined

Summary IBP continity inflow; specified dens and u

9.2.206 Open Flow

Scope: Equation System

Open Flow Flux For Hfem_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary Open bc for continuity

9.2.207 Kinematic

Scope: Equation System

Kinematic Disting For Solid_Y [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for SOLID_y distinguishing condition

9.2.208 Recession

Scope: Equation System

Recession Disting For Solid_Y [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Recession [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* |K = *k* |K_0 = *k_0* |N = *n* |Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for SOLID_x distinguishing condition

9.2.209 Pyrolysis

Scope: Equation System

Pyrolysis Disting For Solid_Y [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Pyrolysis [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* | K = *k* | K_0 = *k_0* | N = *n* | Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for SOLID_y distinguishing condition

9.2.210 Recession

Scope: Equation System

Recession Rotated For Cvfem_Mesh [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Recession [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* | K = *k* | K_0 = *k_0* | N = *n* | Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for rotated CVFEM_MESH boundary condition

9.2.211 Kinematic

Scope: Equation System

Kinematic Rotated For Cvfem_Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for rotated CVFEM_MESH boundary condition

9.2.212 Pyrolysis

Scope: Equation System

Pyrolysis Rotated For Cvfem_Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pyrolysis [Using Data Specification *Data Spec Name*][T_Ref = *t_ref* |K = *k* |K_0 = *k_0* |N = *n* |Temperature = *temperature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for rotated CVFEM_MESH boundary condition

9.2.213 Inflow

Scope: Equation System

Inflow Flux For Cvfem_Solvent_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Inflow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Inflow mass flux for continuity equation; nodal interpolation

9.2.214 Simple Inflow

Scope: Equation System

Simple Inflow Flux For Cvfem_Solvent_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Inflow [Using Data Specification *Data Spec Name*][Density = *Density* |Velocity_X = *Velocity_X* |Velocity_Y = *Velocity_Y* |Velocity_Z = *Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Density	real	undefined
Velocity_X	real	undefined
Velocity_Y	real	undefined
Velocity_Z	real	undefined

Summary Inflow mass flux for continuity equation; specified density and velocity

9.2.215 Open Flow

Scope: Equation System

Open Flow Flux For Cvfem_Solvent_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* | Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary CVFEM solvent momentum open flow flux BC

9.2.216 Kinematic

Scope: Equation System

Kinematic Rotated For Mesh_Colloc [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* | Shared_Normal = *shared_normal* | No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for rotated collocated mesh boundary condition

9.2.217 No Slip

Scope: Equation System

```
No Slip Disting For Mesh [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= No_Slip [ Using Data Specification Data Spec Name ][ V0 = v0 ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined

Summary No slip model for mesh distinguishing condition

9.2.218 Pressure

Scope: Equation System

```
Pressure Flux For Mesh [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Pressure [ Using Data Specification Data Spec Name ][ User_Field_Name = user_field_name
|Use_Bulk_Node = use_bulk_node |P = p |C_T = c_t |C_X = c_x |C_Y = c_y |C_Z = c_z |C_Cos_W
= c_cos_w |C_Sin_W = c_sin_w |C1_Cos = c1_cos |C1_Sin = c1_sin ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
user_field_name	"string"	undefined
use_bulk_node	"string"	undefined
p	real	undefined
c_t	real	undefined
c_x	real	undefined
c_y	real	undefined
c_z	real	undefined
c_cos_w	real	undefined
c_sin_w	real	undefined
c1_cos	real	undefined
c1_sin	real	undefined

Summary Pressure mesh flux

9.2.219 Mixed

Scope: Equation System

Mixed Flux For Spherical_Harmonic [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mixed [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Applies the MIXED scalar flux from the spherical harmonic equations.

9.2.220 Calore User Sub

Scope: Equation System

Calore User Sub Flux For Spherical_Harmonic [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Calore_User_Sub [Using Data Specification *Data Spec Name*][Name = *name* |Type = *type* |Multiplier = *multiplier* |Material_Data_Block = *material_data_block* | Data = *data* |Scaling_Field = *scaling_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
type	"string"	undefined
multiplier	real	undefined
material_data_block	"string"	undefined
data	"string"	undefined
scaling_field	"string"	undefined

Summary Values from a Calore user subroutine

9.2.221 Rad

Scope: Equation System

```
Rad Flux For Enthalpy [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Rad [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |Crad = crad ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
crad	real	undefined

Summary Radiative energy flux defined with constant models for emissivity, form factor and reference temperature

9.2.222 Advection

Scope: Equation System

```
Advection Disting For Level_Set [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Advection [ Using Data Specification Data Spec Name ][ ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Advection model for extension speed distinguishing condition

9.2.223 Open Flow

Scope: Equation System

Open Flow Flux For Mass_Balance [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.224 Mass Open

Scope: Equation System

Mass Open Flux For Mass_Balance [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Mass_Open [Using Data Specification *Data Spec Name*][Pressure = *pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined

9.2.225 Convective Outflow

Scope: Equation System

Convective Outflow Flux For Mass_Balance [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Convective_Outflow [Using Data Specification *Data Spec Name*][Ref_Frac = *ref_frac*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
ref_frac	real	undefined

9.2.226 Simple Interp

Scope: Equation System

Simple Interp Flux For Cvfem_Pressure_Edge_Gradient_Projection [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM edge-based projection simple interpolation flux BC

9.2.227 Open Flow

Scope: Equation System

Open Flow Flux For Cvfem_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* | Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary Open flow for momentum; enforce normal gradient to be zero

9.2.228 Ls Capillary Stabilization

Scope: Equation System

Ls Capillary Stabilization Fluxbp For Cvfem_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExt* | Opposing *OpposingMeshExtent*]= Ls_Capillary_Stabilization [Using Data Specification *Data Spec Name*][Mu0 = *mu0* | Multiplier = *multiplier* | Use_Velocity_Correction = *use_velocity_correction*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC using level set normal

9.2.229 Normal Capillary Stabilization

Scope: Equation System

```
Normal Capillary Stabilization Fluxbp For Cvfem_Momentum [ {of|species|subindex} Species
|{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshEx
|Opposing OpposingMeshExtent ]= Normal_Capillary_Stabilization [ Using Data Specification
Data Spec Name ][ Mu0 = mu0 |Multiplier = multiplier |Use_Velocity_Correction = use_velocity_correctio
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC

9.2.230 Ls Capillary

Scope: Equation System

```
Ls Capillary Fluxbp For Cvfem_Momentum [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Ls_Capillary [ Using Data Specification Data Spec Name ][ ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC using level set normal

9.2.231 Capillary Stabilization

Scope: Equation System

Capillary Stabilization Fluxbp For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Capillary_Stabilization [Using Data Specification *Data Spec Name*][Mu0 = *mu0* |Multiplier = *multiplier* |Use_Velocity_Correction = *use_velocity_correction*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC

9.2.232 Constant Traction

Scope: Equation System

Constant Traction Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Constant_Traction [Using Data Specification *Data Spec Name*][X = *x* |Y = *y* |Z = *z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
x	"string"	undefined
y	"string"	undefined
z	"string"	undefined

9.2.233 Free Open Flow

Scope: Equation System

Free Open Flow Flux For Cvfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Free_Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Pressure User Sub *PRESSURE User Sub*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
PRESSURE User Sub	string	undefined

Summary Open flow for momentum; apply the full stress

9.2.234 Simple Interp

Scope: Equation System

Simple Interp Flux For Hfem_Lumped_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Simple interp model for projection flux

9.2.235 Open

Scope: Equation System

Open Flux For Hfem_Lumped_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open [Using Data Specification *Data Spec Name*][Total_Pressure = *total_pressure* |Pressure = *pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_pressure	real	undefined
pressure	real	undefined

Summary Open model for projection flux

9.2.236 Pressure User Function

Scope: Equation System

Pressure User Function Flux For Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pressure_User_Function [Using Data Specification *Data Spec Name*][Name = *name* |X = *x* |X_Multiplier = *x_multiplier* |Multiplier = *multiplier* |Toggle = *toggle*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
x	"string"	undefined
x_multiplier	real	undefined
multiplier	real	undefined
toggle	"string"	undefined

Summary Pressure user function mesh flux

9.2.237 Wall Function

Scope: Equation System

Wall Function Disting For Cvfem_Specific_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Wall_Function [Using Data Specification *Data Spec Name*][Wall_Friction_Factor = *Wall_Friction_Factor* |Wall_Velocity_X = *Wall_Velocity_X* |Wall_Velocity_Y = *Wall_Velocity_Y* |Wall_Velocity_Z = *Wall_Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Wall_Friction_Factor	real	undefined
Wall_Velocity_X	real	undefined
Wall_Velocity_Y	real	undefined
Wall_Velocity_Z	real	undefined

Summary CVFEM specific dissipation rate distinguishing condition

9.2.238 Low Reynolds

Scope: Equation System

Low Reynolds Disting For Cvfem_Specific_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Low_Reynolds [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.239 Open

Scope: Equation System

Open Flux For Hfem_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open [Using Data Specification *Data Spec Name*][Total_Pressure = *total_pressure* |Pressure = *pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_pressure	real	undefined
pressure	real	undefined

Summary Open model for projection flux

9.2.240 Simple Interp

Scope: Equation System

Simple Interp Flux For Hfem_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Simple interp model for projection flux

9.2.241 Ls Capillary

Scope: Equation System

Ls Capillary Fluxbp For Hfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Ls_Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC using level set normal

9.2.242 Capillary Stabilization

Scope: Equation System

Capillary Stabilization Fluxbp For Hfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Capillary_Stabilization [Using Data Specification *Data Spec Name*][Mu0 = *mu0* |Multiplier = *multiplier* |Use_Velocity_Correction = *use_velocity_correction*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC

9.2.243 Ls Capillary Stabilization

Scope: Equation System

Ls Capillary Stabilization Fluxbp For Hfem_Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Ls_Capillary_Stabilization [Using Data Specification *Data Spec Name*][Mu0 = *mu0* |Multiplier = *multiplier* |Use_Velocity_Correction = *use_velocity_correction*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC using level set normal

9.2.244 Normal Capillary Stabilization

Scope: Equation System

```
   Normal Capillary Stabilization Fluxbp For Hfem_Momentum [ {of|species|subindex} Species
|{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshEx
|Opposing OpposingMeshExtent ]= Normal_Capillary_Stabilization [ Using Data Specification
Data Spec Name ][ Mu0 = mu0 |Multiplier = multiplier |Use_Velocity_Correction = use_velocity_correctio
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mu0	real	undefined
multiplier	real	undefined
use_velocity_correction	integer	undefined

Summary Momentum capillary stabilization flux BP BC

9.2.245 Porous Robin Averaged Coeff

Scope: Equation System

```
   Porous Robin Averaged Coeff Flux For Cvfem_Continuity [ {of|species|subindex} Species |
{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExt
|Opposing OpposingMeshExtent ]= Porous_Robin_Averaged_Coeff [ Using Data Specification Data
Spec Name ][ P_Field = P_Field |Massflux_Field = MassFlux_Field |Oppositecoeff_Field = OppositeCoeff_
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
P_Field	"string"	undefined
MassFlux_Field	"string"	undefined
OppositeCoeff_Field	"string"	undefined

Summary Boundary condition for loose coupling of continuity to a free fluid region

9.2.246 Node Normal Capillary

Scope: Equation System

Node Normal Capillary Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Node_Normal_Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC

9.2.247 Reconstructed Curvature

Scope: Equation System

Reconstructed Curvature Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Reconstructed_Curvature [Using Data Specification *Data Spec Name*][Ls_Size = *ls_size* |Fixed_Curvature = *fixed_curvature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
ls_size	integer	undefined
fixed_curvature	real	undefined

Summary Reconstructed curvature momentum flux

9.2.248 Fixed Curvature

Scope: Equation System

Fixed Curvature Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Fixed_Curvature [Using Data Specification *Data Spec Name*][Curvature = *curvature*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
curvature	real	undefined

Summary Fixed curvature momentum flux

9.2.249 Symmetry Flow

Scope: Equation System

Symmetry Flow Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Symmetry_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Symmetry flow momentum flux

9.2.250 Open Flow Darcy

Scope: Equation System

Open Flow Darcy Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow_Darcy [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary Open flow momentum flux

9.2.251 Exponential Vapor Recoil Pressure

Scope: Equation System

Exponential Vapor Recoil Pressure Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Exponential_Vapor_Recoil_Pressure [Using Data Specification *Data Spec Name*][Tboil = *tboil* |L = *l* |Pa = *pa* |Mw = *mw*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
tboil	real	undefined
l	real	undefined
pa	real	undefined
mw	real	undefined

Summary Exponential vapor recoil pressure momentum flux

9.2.252 Vapor Recoil Pressure

Scope: Equation System

Vapor Recoil Pressure Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Vapor_Recoil_Pressure [Using Data Specification *Data Spec Name*][Tboil = *tboil*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
tboil	real	undefined

Summary Vapor recoil pressure momentum flux

9.2.253 Equilibrium Capillary Gap

Scope: Equation System

Equilibrium Capillary Gap Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Equilibrium_Capillary_Gap [Using Data Specification *Data Spec Name*][Theta = *theta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
theta	real	undefined

Summary Equilibrium gap capillary momentum flux

9.2.254 Capillary Gap

Scope: Equation System

Capillary Gap Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Capillary_Gap [Using Data Specification *Data Spec Name*][V_W = *v_w* | G = *g* | Theta = *theta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v_w	real	undefined
g	real	undefined
theta	real	undefined

Summary Capillary gap momentum flux

9.2.255 Capillary

Scope: Equation System

Capillary Flux For Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary LS capillary momentum flux

9.2.256 Simple Interp

Scope: Equation System

Simple Interp Flux For Cvfem_Sdr_Edge_Gradient_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM edge-based projection simple interpolation flux BC

9.2.257 Inflow

Scope: Equation System

Inflow Flux For Cvfem_Dispersed_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Inflow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Inflow mass flux for dispersed phase continuity equation; nodal interpolation

9.2.258 Open Flow

Scope: Equation System

Open Flow Flux For Cvfem_Dispersed_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Open bc for dispersed continuity

9.2.259 Simple Interp

Scope: Equation System

Simple Interp Flux For Cvfem_Density_Edge_Gradient_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM edge-based projection simple interpolation flux BC

9.2.260 Co2 Convective Outflow

Scope: Equation System

Co2 Convective Outflow Flux For Mass_Balance [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Co2_Convective_Outflow [Using Data Specification *Data Spec Name*][Ref_Frac = *ref_frac*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
ref_frac	real	undefined

9.2.261 Pid Controlled

Scope: Equation System

Pid Controlled Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pid_Controlled [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |P = *p* |I = *i* |D = *d* |Band = *Band* |Filter_Tau = *Filter_Tau* |Setpoint_Function_Name = *Setpoint_Function_Name* |Control_Variable = *Control_Variable* |Max_Output = *Max_Output*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
p	real	undefined
i	real	undefined
d	real	undefined
Band	real	undefined
Filter_Tau	real	undefined
Setpoint_Function_Name	"string"	undefined
Control_Variable	"string"	undefined
Max_Output	real	undefined

9.2.262 Rte Sp

Scope: Equation System

Rte Sp Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Rte_Sp [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Order = *order*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
order	integer	undefined

Summary RTE flux for energy equation with Simplified Spherical Harmonic RTE model

9.2.263 Face Field Scalar

Scope: Equation System

Face Field Scalar Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Face_Field_Scalar [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Name = *name* |Multiplier = *multiplier* |Advective_Bar = *advective_bar* |Command_Block_Name = *command_block_name*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
name	"string"	undefined
multiplier	real	undefined
advective_bar	"string"	undefined
command_block_name	"string"	undefined

Summary FACE or EDGE Field energy flux

9.2.264 Fortran

Scope: Equation System

Fortran Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Fortran [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Multiplier = *multiplier* |Sub_Name = *sub_name* |Real_Data = *real_data* |Int_Data = *int_data* |Resource_Name = *resource_name* |Data = *data*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined
sub_name	"string"	undefined
real_data	"string"	undefined
int_data	"string"	undefined
resource_name	"string"	undefined
data	"string"	undefined

Summary Fortran subroutine energy flux

9.2.265 Masked Nat Conv

Scope: Equation System

Masked Nat Conv Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Masked_Nat_Conv [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Multiplier = *multiplier* |Masking_Field = *masking_field* |Scaling = *scaling* |Threshold = *threshold*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined
masking_field	"string"	undefined
scaling	real	undefined
threshold	real	undefined

Summary Masked natural convection energy flux

9.2.266 Generalized Nat Conv

Scope: Equation System

Generalized Nat Conv Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Generalized_Nat_Conv [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
multiplier	real	undefined

Summary Generalized natural convection energy flux

9.2.267 Rte Field

Scope: Equation System

Rte Field Flux For Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Rte_Field [Using Data Specification *Data Spec Name*][Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined

Summary RTE energy flux from a field

9.2.268 Cht Dirichlet Robin

Scope: Equation System

Cht Dirichlet Robin Flux For Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Cht_Dirichlet_Robin [Using Data Specification *Data Spec Name*][Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Temperature_Field = *temperature_field* | Heat_Flux_Field = *heat_flux_field* | Density_Field = *density_field* | Specific_Heat_Field = *specific_heat_field* | Thermal_Conductivity_Field = *thermal_conductivity_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
temperature_field	"string"	undefined
heat_flux_field	"string"	undefined
density_field	"string"	undefined
specific_heat_field	"string"	undefined
thermal_conductivity_field	"string"	undefined

Summary Applies a Dirichlet-Robin style BC for the energy equation: F_local = F_bc + alpha * (T_local-T_bc), where alpha is computed based on a 1D analytical solution.

9.2.269 Cht Robin

Scope: Equation System

Cht Robin Flux For Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Cht_Robin [Using Data Specification *Data Spec Name*][Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Temperature_Field = *temperature_field* | Heat_Flux_Field = *heat_flux_field* | Coeff_Scaling = *coeff_scaling*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
temperature_field	"string"	undefined
heat_flux_field	"string"	undefined
coeff_scaling	real	undefined

Summary Applies a Robin-style BC for the energy equation: F_local = F_bc + alpha * (T_local-T_bc).

9.2.270 Cht Flux

Scope: Equation System

Cht Flux Flux For Energy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Cht_Flux [Using Data Specification *Data Spec Name*][Power_Output = *power_output* | Flux_Output = *flux_output* | Toggle = *toggle* | Too_Field = *Too_Field* | H_Field = *H_Field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
Too_Field	"string"	undefined
H_Field	"string"	undefined

Summary Applies H(T-Too) flux for conjugate heat transfer boundary. Optional parameters are "Too_Field" and "H_Field", which default to "nodal_cht_Too" and "nodal_cht_H" respectively.

9.2.271 Transient Traction

Scope: Equation System

Transient Traction Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Transient_Traction [Using Data Specification *Data Spec Name*][A_X = *a_x* |A_Y = *a_y* |A_Z = *a_z* |B_X = *b_x* |B_Y = *b_y* |B_Z = *b_z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
a_x	real	undefined
a_y	real	undefined
a_z	real	undefined
b_x	real	undefined
b_y	real	undefined
b_z	real	undefined

9.2.272 Pressure User Function

Scope: Equation System

Pressure User Function Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pressure_User_Function [Using Data Specification *Data Spec Name*][Name = *name* |X = *x* |X_Multiplier = *x_multiplier* |Multiplier = *multiplier* |Toggle = *toggle*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
name	"string"	undefined
x	"string"	undefined
x_multiplier	real	undefined
multiplier	real	undefined
toggle	"string"	undefined

Summary Pressure user function momentum flux

9.2.273 Pressure Darcy

Scope: Equation System

Pressure Darcy Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase}

MaterialPhase]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pressure_Darcy [Using Data Specification *Data Spec Name*][User_Field_Name = *user_field_name* |Use_Bulk_Node = *use_bulk_node* |P = *p* |C_T = *c_t* |C_X = *c_x* |C_Y = *c_y* |C_Z = *c_z* |C_Cos_W = *c_cos_w* |C_Sin_W = *c_sin_w* |C1_Cos = *c1_cos* |C1_Sin = *c1_sin*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
user_field_name	"string"	undefined
use_bulk_node	"string"	undefined
p	real	undefined
c_t	real	undefined
c_x	real	undefined
c_y	real	undefined
c_z	real	undefined
c_cos_w	real	undefined
c_sin_w	real	undefined
c1_cos	real	undefined
c1_sin	real	undefined

9.2.274 Constant Traction

Scope: Equation System

Constant Traction Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Constant_Traction [Using Data Specification *Data Spec Name*][X = *x* |Y = *y* |Z = *z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
x	"string"	undefined
y	"string"	undefined
z	"string"	undefined

9.2.275 Free Open Flow

Scope: Equation System

Free Open Flow Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Free_Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Pressure User Sub *PRESSURE User Sub*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
PRESSURE User Sub	string	undefined

Summary Open flow for momentum; apply the full stress

9.2.276 Pressure

Scope: Equation System

Pressure Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pressure [Using Data Specification *Data Spec Name*][User_Field_Name = *user_field_name* |Use_Bulk_Node = *use_bulk_node* |P = *p* |C_T = *c_t* |C_X = *c_x* |C_Y = *c_y* |C_Z = *c_z* |C_Cos_W = *c_cos_w* |C_Sin_W = *c_sin_w* |C1_Cos = *c1_cos* |C1_Sin = *c1_sin*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
user_field_name	"string"	undefined
use_bulk_node	"string"	undefined
p	real	undefined
c_t	real	undefined
c_x	real	undefined
c_y	real	undefined
c_z	real	undefined
c_cos_w	real	undefined
c_sin_w	real	undefined
c1_cos	real	undefined
c1_sin	real	undefined

Summary Pressure momentum flux

9.2.277 Open Flow

Scope: Equation System

Open Flow Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary Open flow for momentum; enforce normal gradient to be zero

9.2.278 Non Ibp Pressure

Scope: Equation System

Non Ibp Pressure Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Non_Ibp_Pressure [Using Data Specification *Data Spec Name*][Pressure = *pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	"string"	undefined

9.2.279 Rt Pressure Darcy

Scope: Equation System

Rt Pressure Darcy Flux For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Rt_Pressure_Darcy [Using Data Specification *Data Spec Name*][P = *p* |C_T = *c_t* |C_X = *c_x* |C_Y = *c_y* |C_Z = *c_z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
p	real	undefined
c_t	real	undefined
c_x	real	undefined
c_y	real	undefined
c_z	real	undefined

9.2.280 Open Adv Flow

Scope: Equation System

```
Open Adv Flow Flux For Hfem_Mixture_Fraction [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Open_Adv_Flow [ Using Data Specification Data Spec Name ][ Pressure
= pressure |Total_Pressure = total_pressure |Far_Field_Entrainment_Value = far_field_entrainment_value
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

Summary HFEM mixture fraction open advective flow flux BC

9.2.281 Vector User Function Disting

Scope: Equation System

```
Vector User Function Disting Disting For Momentum [ {of|species|subindex} Species |{in
|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent
|Opposing OpposingMeshExtent ]= Vector_User_Function_Disting [ Using Data Specification Data
Spec Name ][ Name_X = name_x |Name_Y = name_y |Name_Z = name_z |X = x |Multiplier = multiplier
|Mult = Mult ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
name_x	"string"	undefined
name_y	"string"	undefined
name_z	"string"	undefined
x	"string"	undefined
multiplier	real	undefined
Mult	real	undefined

Summary Vector user function model for momentum distinguishing condition

9.2.282 Electroosmotic Velocity

Scope: Equation System

Electroosmotic Velocity Disting For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Electroosmotic_Velocity [Using Data Specification *Data Spec Name*][C = *c*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
c	real	undefined

Summary Electro-osmotic velocity model for momentum distinguishing condition

9.2.283 Wetting Speed Blake Ls

Scope: Equation System

Wetting Speed Blake Ls Disting For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Wetting_Speed_Blake_Ls [Using Data Specification *Data Spec Name*][V_W = *v_w* |G = *g* |Theta = *theta* |Width = *width* |Tau = *tau*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v_w	real	undefined
g	real	undefined
theta	real	undefined
width	real	undefined
tau	real	undefined

Summary Blake level set wetting speed model for momentum distinguishing condition

9.2.284 No Slip

Scope: Equation System

No Slip Disting For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= No_Slip [Using Data Specification *Data Spec Name*][V0 = *v0*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined

Summary No slip model for momentum distinguishing condition

9.2.285 Wetting

Scope: Equation System

Wetting Edge For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]Where *SidePart0Name* Intersects *SidePart1Name* = Wetting [Theta_S = *theta_s*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
SidePart0Name	string	undefined
SidePart1Name	string	undefined
theta_s	real	undefined

Summary Sharp wetting edge source term

9.2.286 Darcy Leak

Scope: Equation System

Darcy Leak Disting For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Darcy_Leak [Using Data Specification *Data Spec Name*][P_Ref = *p_ref* |L = *l* |K = *k* | Pressure = *pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
p_ref	real	undefined
l	real	undefined
k	real	undefined
pressure	"string"	undefined

Summary Darcy leak model for momentum distinguishing condition

9.2.287 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Mass_Fraction [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Total_Mdot = *total_mdot* |Inflow_Phi = *inflow_phi*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_mdot	real	undefined
inflow_phi	real	undefined

9.2.288 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Cvfem_Mass_Fraction [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure* |Far_Field_Entrainment_Value = *far_field_entrainment_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

9.2.289 Porous Robin Coupled

Scope: Equation System

Porous Robin Coupled Flux For Cvfem_Mass_Fraction [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Porous_Robin_Coupled [Using Data Specification *Data Spec Name*][Mass_Fraction_Field = *mass_fraction_field* |Mass_Fraction_Diffusive_Flux_Field = *mass_fraction_* |Averaged_Coeff_Field = *averaged_coeff_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
mass_fraction_field	"string"	undefined
mass_fraction_ diffusive_flux_field	"string"	undefined
averaged_ coeff_field	"string"	undefined

Summary Boundary condition for loose coupling of scalars to a free fluid region

9.2.290 Porous Robin Coupled One Region

Scope: Equation System

Porous Robin Coupled One Region Flux For Cvfem_Mass_Fraction [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshEx* | Opposing *OpposingMeshExtent*]= Porous_Robin_Coupled_One_Region [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Boundary condition for loose coupling of scalars to a free fluid region

9.2.291 Open Flow

Scope: Equation System

Open Flow Flux For Cvfem_Solvent_Continuity [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* | Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_ pressure	real	undefined

Summary Open bc for continuity

9.2.292 Recession

Scope: Equation System

```
Recession Disting For Mesh_Z [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Recession [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for mesh_x distinguishing condition

9.2.293 Kinematic

Scope: Equation System

```
Kinematic Disting For Mesh_Z [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Kinematic [ Using Data Specification Data Spec Name ][ V0 = v0 |Shared_Normal = shared_normal
|No_Mesh_Movement = no_mesh_movement ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for mesh_z distinguishing condition

9.2.294 Inflow

Scope: Equation System

```
Inflow Flux For Cvfem_Continuity [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Inflow [ Using Data Specification Data Spec Name ][ ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary IBP continity inflow; nodal interp

9.2.295 Open Flow

Scope: Equation System

Open Flow Flux For Cvfem_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* |Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary Open bc for continuity

9.2.296 Simple Inflow

Scope: Equation System

Simple Inflow Flux For Cvfem_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Inflow [Using Data Specification *Data Spec Name*][Density = *Density* |Velocity_X = *Velocity_X* |Velocity_Y = *Velocity_Y* |Velocity_Z = *Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Density	real	undefined
Velocity_X	real	undefined
Velocity_Y	real	undefined
Velocity_Z	real	undefined

Summary IBP continity inflow; specified dens and u

9.2.297 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Total_Mdot = *total_mdot*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_mdot	real	undefined

Summary Mass flux CVFEM continuity flux

9.2.298 Open Adv Pen Flow

Scope: Equation System

Open Adv Pen Flow Flux For Cvfem_Level_Set [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Adv_Pen_Flow [Using Data Specification *Data Spec Name*][Penalty_Factor = *penalty_factor* |Far_Field_Entrainment_Value = *far_field_entrainment_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
penalty_factor	real	undefined
far_field_entrainment_value	real	undefined

9.2.299 Open Nc Adv Flow

Scope: Equation System

Open Nc Adv Flow Flux For Cvfem_Level_Set [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open_Nc_Adv_Flow [Using Data Specification *Data Spec Name*][Far_Field_Entrainme = *far_field_entrainment_value*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
far_field_entrainment_value	real	undefined

9.2.300 Open Adv Flow

Scope: Equation System

```
  Open Adv Flow Flux For Cvfem_Level_Set [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Open_Adv_Flow [ Using Data Specification Data Spec Name ][ Pressure
= pressure |Total_Pressure = total_pressure |Far_Field_Entrainment_Value = far_field_entrainment_value
]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

9.2.301 Simple Interp Tensor

Scope: Equation System

```
  Simple Interp Tensor Flux For Cvfem_Lumped_Div_Projection [ {of|species|subindex} Species
|{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshEx
|Opposing OpposingMeshExtent ]= Simple_Interp_Tensor [ Using Data Specification Data Spec
Name ][ Source = source ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
source	"string"	undefined

Summary CVFEM lumped divergence projection simple interpolation tensor flux BC

9.2.302 Generalized Nat Conv

Scope: Equation System

Generalized Nat Conv No_Coverage_Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Generalized_Nat_Conv [Using Data Specification *Data Spec Name*][Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
multiplier	real	undefined

Summary Generalized natural convection energy flux

9.2.303 Constant Velocity

Scope: Equation System

Constant Velocity Flux For Mass_Balance [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Constant_Velocity [Using Data Specification *Data Spec Name*][Mult = *Mult*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Mult	real	undefined

9.2.304 Lens

Scope: Equation System

Lens Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Lens [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Src_X = *src_x* |Dir_X = *dir_x* |Vel_X = *vel_x* |Src_Y = *src_y* |Dir_Y = *dir_y* |Vel_Y = *vel_y* |Src_Z = *src_z* |Dir_Z = *dir_z* |Vel_Z = *vel_z* |Massflowrate = *massFlowRate* |Particlespeed = *particleSpeed* |Particledensity = *particleDensity* |Masssourceradius = *massSourceRadius* |T_Liq = *T_liq* |T_Solid = *T_solid* |Particletemperature = *particleTemperature* |T_On = *t_on* |T_Off = *t_off*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
src_x	real	undefined
dir_x	real	undefined
vel_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
vel_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
vel_z	real	undefined
massFlowRate	real	undefined
particleSpeed	real	undefined
particleDensity	real	undefined
massSourceRadius	real	undefined
T_liq	real	undefined
T_solid	real	undefined
particleTemperature	real	undefined
t_on	real	undefined
t_off	real	undefined

9.2.305 Sharp Line Weld

Scope: Equation System

```
   Sharp Line Weld Flux For Energy [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Sharp_Line_Weld [ Using Data Specification Data Spec Name ][ Power_Output = power_output
|Flux_Output = flux_output |Toggle = toggle |Src_X = src_x |Dir_X = dir_x |Vel_X = vel_x
|Src_Y = src_y |Dir_Y = dir_y |Vel_Y = vel_y |Src_Z = src_z |Dir_Z = dir_z |Vel_Z = vel_z
|R = r |T_On = t_on |T_Off = t_off |Flux = flux |Alpha = alpha |Depth_Enhanced_Absorption_Data
= depth_enhanced_absorption_data |Compute_Visibility_Field = compute_visibility_field ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
src_x	real	undefined
dir_x	real	undefined
vel_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
vel_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
vel_z	real	undefined
r	real	undefined
t_on	real	undefined
t_off	real	undefined
flux	real	undefined
alpha	real	undefined
depth_enhanced_absorption_data	"string"	undefined
compute_visibility_field	"string"	undefined

9.2.306 Gaussian Line Weld

Scope: Equation System

Gaussian Line Weld Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Gaussian_Line_Weld [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Src_X = *src_x* |Dir_X = *dir_x* |Vel_X = *vel_x* |Src_Y = *src_y* |Dir_Y = *dir_y* |Vel_Y = *vel_y* |Src_Z = *src_z* |Dir_Z = *dir_z* |Vel_Z = *vel_z* |R = *r* |T_On = *t_on* |T_Off = *t_off* |Flux = *flux* |R_Eff = *r_eff* |Alpha = *alpha* |Depth_Enhanced_Absorption_Data = *depth_enhanced_absorption_data* |Compute_Visibility_Field = *compute_visibility_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
src_x	real	undefined
dir_x	real	undefined
vel_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
vel_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
vel_z	real	undefined
r	real	undefined
t_on	real	undefined
t_off	real	undefined
flux	real	undefined
r_eff	real	undefined
alpha	real	undefined
depth_enhanced_absorption_data	"string"	undefined
compute_visibility_field	"string"	undefined

9.2.307 Enclosure Radiation

Scope: Equation System

Enclosure Radiation Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Enclosure_Radiation [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Ndof = *ndof* |Multiplier = *multiplier* |Enclosure = *enclosure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
ndof	integer	undefined
multiplier	real	undefined
enclosure	"string"	undefined

9.2.308 Gaussian Spot Weld

Scope: Equation System

Gaussian Spot Weld Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Gaussian_Spot_Weld [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |Src_X = *src_x* |Dir_X = *dir_x* |Src_Y = *src_y* |Dir_Y = *dir_y* |Src_Z = *src_z* |Dir_Z = *dir_z* |R = *r* |T_On = *t_on* |T_Off = *t_off* |Flux = *flux* |R_Eff = *r_eff* |Alpha = *alpha* |Depth_Enhanced_Absorption_Data = *depth_enhanced_* |Compute_Visibility_Field = *compute_visibility_field*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
src_x	real	undefined
dir_x	real	undefined
src_y	real	undefined
dir_y	real	undefined
src_z	real	undefined
dir_z	real	undefined
r	real	undefined
t_on	real	undefined
t_off	real	undefined
flux	real	undefined
r_eff	real	undefined
alpha	real	undefined
depth_enhanced_absorption_data	"string"	undefined
compute_visibility_field	"string"	undefined

9.2.309 Rad

Scope: Equation System

Rad Flux For Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Rad [Using Data Specification *Data Spec Name*][Power_Output = *power_output* |Flux_Output = *flux_output* |Toggle = *toggle* |T_Ref = *t_ref* |Crad = *crad*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
power_output	"string"	undefined
flux_output	"string"	undefined
toggle	"string"	undefined
t_ref	real	undefined
crad	real	undefined

Summary Radiative energy flux defined with constant models for emissivity, form factor and reference temperature

9.2.310 Wall Function From Cht Temperature

Scope: Equation System

```
Wall Function From Cht Temperature Flux For Cvfem_Energy [ {of|species|subindex} Species
|{in|material_phase} MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshEx
|Opposing OpposingMeshExtent ]= Wall_Function_From_Cht_Temperature [ Using Data Specification
Data Spec Name ][ Data = data |Wall_Friction_Factor = wall_friction_factor ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
wall_friction_factor	real	undefined

Summary CVFEM energy wall function from conjugate heat transfer temperature flux BC

9.2.311 Porous Robin Coupled

Scope: Equation System

```
Porous Robin Coupled Flux For Cvfem_Energy [ {of|species|subindex} Species |{in|material_phase}
MaterialPhase ]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing
OpposingMeshExtent ]= Porous_Robin_Coupled [ Using Data Specification Data Spec Name ][ Data
= data |Enthalpy_Field = enthalpy_field |Enthalpy_Diffusive_Flux_Field = enthalpy_diffusive_flux_field
|Averaged_Coeff_Field = averaged_coeff_field ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
enthalpy_field	"string"	undefined
enthalpy_diffusive_flux_field	"string"	undefined
averaged_coeff_field	"string"	undefined

Summary Boundary condition for loose coupling of enthalpy to a porous region

9.2.312 Convective Outflow

Scope: Equation System

Convective Outflow Flux For Enthalpy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Convective_Outflow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.313 Generalized Rad

Scope: Equation System

Generalized Rad Flux For Enthalpy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Generalized_Rad [Using Data Specification *Data Spec Name*][Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
multiplier	real	undefined

Summary Radiative energy flux, possibly defined with different models for emissivity, form factor and irradiation

9.2.314 Generalized Nat Conv

Scope: Equation System

Generalized Nat Conv Flux For Enthalpy [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Generalized_Nat_Conv [Using Data Specification *Data Spec Name*][Multiplier = *multiplier*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
multiplier	real	undefined

Summary Generalized natural convection enthalpy flux

9.2.315 Open Flow

Scope: Equation System

Open Flow Flux For Hfem_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Open_Flow [Using Data Specification *Data Spec Name*][Pressure = *pressure* | Total_Pressure = *total_pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined

Summary Open flow for momentum; enforce normal gradient to be zero

9.2.316 Balanced Force

Scope: Equation System

Balanced Force Flux For Hfem_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Balanced_Force [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary HFEM momentum balanced force flux BC

9.2.317 Symmetry Flow

Scope: Equation System

Symmetry Flow Flux For Hfem_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Symmetry_Flow [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary HFEM momentum symmetry flow flux BC

9.2.318 Interface

Scope: Equation System

Interface Flux For Hfem_Momentum [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshExtent* | Opposing *OpposingMeshExtent*]= Interface [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary INTERFACE HFEM momentum flux

9.2.319 Free Open Flow

Scope: Equation System

Free Open Flow Flux For Cvfem_Turbulence_Dissipation_Rate [{of | species | subindex} *Species* | {in | material_phase} *MaterialPhase*]{@ | at | for | in | on | over} *Mesh Extent Name* [Touching *TouchingMeshEx*

|Opposing *OpposingMeshExtent*]= Free_Open_Flow [Using Data Specification *Data Spec Name*][
]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

9.2.320 Wall Function

Scope: Equation System

Wall Function Disting For Cvfem_Turbulent_Kinetic_Energy [{of|species|subindex} *Species*
|{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx*
|Opposing *OpposingMeshExtent*]= Wall_Function [Using Data Specification *Data Spec Name*][
Wall_Friction_Factor = *Wall_Friction_Factor* |Wall_Velocity_X = *Wall_Velocity_X* |Wall_Velocity_Y
= *Wall_Velocity_Y* |Wall_Velocity_Z = *Wall_Velocity_Z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
Wall_Friction_Factor	real	undefined
Wall_Velocity_X	real	undefined
Wall_Velocity_Y	real	undefined
Wall_Velocity_Z	real	undefined

Summary CVFEM wall function turbulent kinetic energy distinguishing condition

9.2.321 Open Adv Flow

Scope: Equation System

Open Adv Flow Flux For Cvfem_Turbulence_Dissipation_Rate [{of|species|subindex} *Species*
|{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx*
|Opposing *OpposingMeshExtent*]= Open_Adv_Flow [Using Data Specification *Data Spec Name*][
Pressure = *pressure* |Total_Pressure = *total_pressure* |Far_Field_Entrainment_Value = *far_field_entrainm*
]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
pressure	real	undefined
total_pressure	real	undefined
far_field_entrainment_value	real	undefined

9.2.322 Mass Flux

Scope: Equation System

Mass Flux Flux For Cvfem_Turbulence_Dissipation_Rate [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExt* |Opposing *OpposingMeshExtent*]= Mass_Flux [Using Data Specification *Data Spec Name*][Total_Mdot = *total_mdot* |Inflow_Phi = *inflow_phi*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_mdot	real	undefined
inflow_phi	real	undefined

9.2.323 Smoothed Capillary

Scope: Equation System

Smoothed Capillary Fluxbp For Momentum [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Smoothed_Capillary [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Momentum capillary flux BP BC

9.2.324 Pspg

Scope: Equation System

Pspg Fluxbp For Continuity [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Pspg [Using Data Specification *Data Spec Name*][Tausurfacefactor = *TauSurfaceFactor* | Tausurfacejacobianfactor = *TauSurfaceJacobianFactor*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
TauSurfaceFactor	real	undefined
TauSurfaceJacobianFactor	real	undefined

Summary Continuity PSPG flux BP BC

9.2.325 Kinematic

Scope: Equation System

Kinematic Disting For Mesh_Y [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for mesh_y distinguishing condition

9.2.326 Simple Interp

Scope: Equation System

Simple Interp Flux For Hfem_Bf_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Simple interp model for BF projection flux

9.2.327 From Temperature

Scope: Equation System

From Temperature Disting For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= From_Temperature [Using Data Specification *Data Spec Name*][Variable = *variable* |Order = *order* |Variable_Offset = *variable_offset* |C0 = *c0* |C1 = *c1* |C2 = *c2* |C3 = *c3* |C4 = *c4* |C5 = *c5* |C6 = *c6* |C7 = *c7* |C8 = *c8* |T = *t* |Poffset = *poffset*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
variable	"string"	undefined
order	integer	undefined
variable_offset	real	undefined
c0	real	undefined
c1	real	undefined
c2	real	undefined
c3	real	undefined
c4	real	undefined
c5	real	undefined
c6	real	undefined
c7	real	undefined
c8	real	undefined
t	real	undefined
poffset	real	undefined

Summary From temperature distinguishing condition for CVFEM energy equation

9.2.328 Wall Function From Temperature

Scope: Equation System

Wall Function From Temperature Flux For Cvfem_Energy [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExt* |Opposing *OpposingMeshExtent*]= Wall_Function_From_Temperature [Using Data Specification *Data Spec Name*][Data = *data* |T = *t* |Wall_Friction_Factor = *wall_friction_factor*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
data	"string"	undefined
t	real	undefined
wall_friction_factor	real	undefined

Summary CVFEM energy wall function from temperature flux BC

9.2.329 Simple Dp Interp

Scope: Equation System

Simple Dp Interp Flux For Cvfem_Velocity_Pressure_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshEx* |Opposing *OpposingMeshExtent*]= Simple_Dp_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Simple DP interploation model for HFEM velocity pressure projecion flux

9.2.330 Simple Interp

Scope: Equation System

Simple Interp Flux For Cvfem_Velocity_Edge_Gradient_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Simple_Interp [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary CVFEM edge-based projection simple interpolation flux BC

9.2.331 Pyrolysis

Scope: Equation System

```
  Pyrolysis Rotated For Mesh [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Pyrolysis [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Pyrolysis model for rotated mesh boundary condition

9.2.332 Recession

Scope: Equation System

```
  Recession Rotated For Mesh [ {of|species|subindex} Species |{in|material_phase} MaterialPhase
]{@|at|for|in|on|over} Mesh Extent Name [ Touching TouchingMeshExtent |Opposing OpposingMeshExtent
]= Recession [ Using Data Specification Data Spec Name ][ T_Ref = t_ref |K = k |K_0 = k_0
|N = n |Temperature = temperature ]
```

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
t_ref	real	undefined
k	real	undefined
k_0	real	undefined
n	real	undefined
temperature	"string"	undefined

Summary Recession model for rotated mesh boundary condition

9.2.333 Shear Free

Scope: Equation System

Shear Free Flux For Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Shear_Free [Using Data Specification *Data Spec Name*][ ]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined

Summary Shear-free traction mesh flux

9.2.334 Kinematic

Scope: Equation System

Kinematic Rotated For Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Kinematic [Using Data Specification *Data Spec Name*][V0 = *v0* |Shared_Normal = *shared_normal* |No_Mesh_Movement = *no_mesh_movement*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
v0	real	undefined
shared_normal	"string"	undefined
no_mesh_movement	"string"	undefined

Summary Kinematic model for rotated mesh boundary condition

9.2.335 Electric Traction

Scope: Equation System

Electric Traction Flux For Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Electric_Traction [Using Data Specification *Data Spec Name*][Sign = *sign*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
sign	real	undefined

Summary Electric traction mesh flux

9.2.336 Constant Traction

Scope: Equation System

Constant Traction Flux For Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Constant_Traction [Using Data Specification *Data Spec Name*][X = *x* |Y = *y* |Z = *z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
x	"string"	undefined
y	"string"	undefined
z	"string"	undefined

Summary Constant traction mesh flux

9.2.337 Transient Traction

Scope: Equation System

Transient Traction Flux For Mesh [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Transient_Traction [Using Data Specification *Data Spec Name*][A_X = *a_x* |A_Y = *a_y* |A_Z = *a_z* |B_X = *b_x* |B_Y = *b_y* |B_Z = *b_z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
a_x	real	undefined
a_y	real	undefined
a_z	real	undefined
b_x	real	undefined
b_y	real	undefined
b_z	real	undefined

Summary Transient traction mesh flux

9.2.338 Capillary Force

Scope: Equation System

Capillary Force Flux For Lumped_Div_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Capillary_Force [Using Data Specification *Data Spec Name*][Theta = *theta*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
theta	real	undefined

Summary Capillary force model for lumped div projection flux

9.2.339 Open

Scope: Equation System

Open Flux For Cvfem_Projection [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Open [Using Data Specification *Data Spec Name*][Total_Pressure = *total_pressure* |Pressure = *pressure*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
total_pressure	real	undefined
pressure	real	undefined

Summary CVFEM projection open flux BC

9.2.340 Transient Traction

Scope: Equation System

Transient Traction Flux For Solid [{of|species|subindex} *Species* |{in|material_phase} *MaterialPhase*]{@|at|for|in|on|over} *Mesh Extent Name* [Touching *TouchingMeshExtent* |Opposing *OpposingMeshExtent*]= Transient_Traction [Using Data Specification *Data Spec Name*][A_X = *a_x* |A_Y = *a_y* |A_Z = *a_z* |B_X = *b_x* |B_Y = *b_y* |B_Z = *b_z*]

Parameter	Value	Default
Species	string	undefined
MaterialPhase	string	undefined
Mesh Extent Name	string	undefined
TouchingMeshExtent	string	undefined
OpposingMeshExtent	string	undefined
Data Spec Name	string	undefined
a_x	real	undefined
a_y	real	undefined
a_z	real	undefined
b_x	real	undefined
b_y	real	undefined
b_z	real	undefined

Summary Transient traction solid flux

9.2.341 Bc Bulk Node Coupling For Density = K Factor Flow

Scope: Equation System

```
Bc Bulk Node Coupling For Density = K Factor Flow [ Flow_Rate_Limiter = flow_rate_limiter
]
```

Parameter	Value	Default
flow_rate_limiter	real	undefined

Summary Volumetric flow rate, Q, calculated as $Q = K\sqrt{\frac{P_{high}-P_{low}}{\rho}}$. Absent any other sources, the maximum flow rate is defined based on the mass loss to achieve pressure equilibrium and the fluid time step. If the optional flow rate limiter is specified, the flow rate is limited by the product of the limiter factor and the maximum flow rate without any other sources.

9.2.342 Bc Bulk Node Coupling For Density = K Factor Flow With Choking

Scope: Equation System

```
Bc Bulk Node Coupling For Density = K Factor Flow With Choking [ Critical_Pressure_Ratio
= critical_pressure_ratio |Flow_Rate_Limiter = flow_rate_limiter ]
```

Parameter	Value	Default
critical_pressure_ratio	real	undefined
flow_rate_limiter	real	undefined

Summary Volumetric flow rate, Q, calculated as $Q = K\sqrt{\frac{(P_{high}-P_{low})}{\rho}}$ for $\frac{P_{high}}{P_{low}} < R_p$, where R_p is the critical_pressure_ratio. Above R_p the flow rate is calculated as $Q = K * \sqrt{\frac{P_{high}(1-1/R_p)}{\rho}}$. Absent any other sources, the maximum flow rate is defined based on the mass loss to achieve pressure equilibrium and the fluid time step. If the optional flow rate limiter is specified, the flow rate is limited by the product of the limiter factor and the maximum flow rate without any other sources.

9.2.343 Bc Bulk Node Coupling For Species = K Factor Flow

Scope: Equation System

`Bc Bulk Node Coupling For Species = K Factor Flow [ Flow_Rate_Limiter =` *flow_rate_limiter* `]`

Parameter	Value	Default
flow_rate_limiter	`real`	`undefined`

Summary Volumetric flow rate, Q, calculated as $Q = K\sqrt{\frac{P_{high}-P_{low}}{\rho}}$. Absent any other sources, the maximum flow rate is defined based on the mass loss to achieve pressure equilibrium and the fluid time step. If the optional flow rate limiter is specified, the flow rate is limited by the product of the limiter factor and the maximum flow rate without any other sources.

9.2.344 Bc Bulk Node Coupling For Species = K Factor Flow With Choking

Scope: Equation System

`Bc Bulk Node Coupling For Species = K Factor Flow With Choking [ Critical_Pressure_Ratio =` *critical_pressure_ratio* `|Flow_Rate_Limiter =` *flow_rate_limiter* `]`

Parameter	Value	Default
critical_pressure_ratio	`real`	`undefined`
flow_rate_limiter	`real`	`undefined`

Summary Volumetric flow rate, Q, calculated as $Q = K\sqrt{\frac{(P_{high}-P_{low})}{\rho}}$ for $\frac{P_{high}}{P_{low}} < R_p$, where R_p is the critical_pressure_ratio. Above R_p the flow rate is calculated as $Q = K * \sqrt{\frac{P_{high}(1-1/R_p)}{\rho}}$. Absent any other sources, the maximum flow rate is defined based on the mass loss to achieve pressure equilibrium and the fluid time step. If the optional flow rate limiter is specified, the flow rate is limited by the product of the limiter factor and the maximum flow rate without any other sources.

9.2.345 Bc Interface Colloc Disting For Lubrication Height = Deforming

Scope: Equation System

`Bc Interface Colloc Disting For Lubrication Height = Deforming [ Hmin =` *hmin* `]`

Parameter	Value	Default
hmin	`real`	`undefined`

Summary Collocated distinguishing condition for the lubrication height that deforms by the displacement of the adjoining volume region.

9.3 Periodic BC Overview

Aria provides a periodic boundary condition capability. Two flavors of periodicity are supported, standard periodicity for surfaces separated by fixed distance and cyclic periodicity for surfaces separated through a

fixed rotation angle as illustrated in Figure 9.1. The underlying implementation is based upon augmenting the basic governing equations with multi-point constraints on nodal DOF at the periodic boundaries and is equally applicable to scalar and vector DOF. The enforcement employs nodal DOF, hence the capability is limited to grids with conforming surface nodes, i.e. the mesh is periodic as well.

Internally, a search procedure is used in order to define master-slave node constraint pairs. Since a meshed discretization is subject to intrinsic error in the nodal coordinates, an acceptable tolerance specification is used to insure that master-slave pairings are appropriately defined. Note that application of large tolerances may lead to incorrectly defined master-slave pairings. Additional quantities required for cyclic periodicity are referenced in 9.4.7.

Figure 9.1. Standard and Cyclic Periodicity

9.4 Periodic

Scope: Aria Region

```
Begin Periodic Name
    Master {= | are | is} Master
    Point On Axis {= | are | is} Point
    Reference Axis {= | are | is} Axis
    Search Tolerance {= | are | is} SearchTolerance
    Slave {= | are | is} Slave
    Theta {= | are | is} Theta
End
```

Summary — The periodic command block is used to define periodic, or cyclic, boundary conditions.

Description — For standard periodic boundary conditions, the line commands MASTER, SLAVE, SEARCH TOLERANCE should appear, while the line commands THETA, POINT ON AXIS, and REFERENCE AXIS must not appear.

Cyclic boundary conditions, the line commands MASTER, SLAVE, SEARCH TOLERANCE, THETA, POINT ON AXIS, and REFERENCE AXIS must appear.

Two self-similar side sets for the boundary condition are specified by the MASTER and SLAVE command lines. It must be possible to replicate the coordinates for the nodes in one set by a simple translation of the corresponding nodes in the other set.

A geometric check is made to determine if the side sets are self similar. The geometric check is set to a certain tolerance. This tolerance must be set to a small positive number (e.g. 10^{-4}) by using the command line SEARCH TOLERANCE. After translation each node in the set of slave nodes lies within a radius defined by the search tolerance from a node in the set of master nodes. Additionally, upon transformation, slave nodes must lie in the interpolation space of at least one facet in the master surface. When these two conditions are satisfied the two sets are considered self-similar.

9.4.1 Master

Scope: Periodic

`Master {= | are | is}` *`Master`*

Parameter	Value	Default
Master	`string`	`undefined`

Summary Defines master surface for this interaction. When used in Aria and Calore, this must always be a sideset.

9.4.2 Point On Axis

Scope: Periodic

`Point On Axis {= | are | is}` *`Point`*

Parameter	Value	Default
Point	`string`	`undefined`

Summary This line command defines a point that lies on the reference axis.

9.4.3 Reference Axis

Scope: Periodic

`Reference Axis {= | are | is}` *`Axis`*

Parameter	Value	Default
Axis	`string`	`undefined`

Summary This line command defines the reference wedge axis for the periodic BC about which the master surface is rotated to transform it into the slave surface.

9.4.4 Search Tolerance

Scope: Periodic

`Search Tolerance {=|are|is} SearchTolerance`

Parameter	Value	Default
SearchTolerance	`real`	`undefined`

Summary This line command defines the search tolerance used to determine if two node sets are self similar.

9.4.5 Slave

Scope: Periodic

`Slave {=|are|is} Slave`

Parameter	Value	Default
Slave	`string`	`undefined`

Summary Defines slave surface for this interaction. When used in Aria and Calore, this may be a nodeset or a sideset.

9.4.6 Theta

Scope: Periodic

`Theta {=|are|is} Theta`

Parameter	Value	Default
Theta	`real`	`undefined`

Summary This line command defines the angle, in degrees, for a periodic boundary condition. A positive angle is defined by sweeping the master surface through the domain to the slave surface.

9.4.7 Cyclic Periocic BC Parameters

Cyclic periodicity requires that the user specify both **Reference Axis** of rotation and **Point on Axis** (a point on the reference axis) in the Begin Periodic command block. Here Reference Axis and Point on Axis are associated with **Define Direction** and **Define Point** respectively. These two additional quantities are usually defined at the Domain scope (outside of the Procedure).

9.4.8 Define Direction

Scope:

`Define Direction DirectName With Vector` $Components_1$ $Components_2$ $Components_3$

Parameter	Value	Default
DirectName	`string`	`undefined`
Components	`real_1 real_2 real_3`	`undefined`

Summary Defines a named spatial direction in terms of vector components.

9.4.9 Define Point

Scope:

`Define Point` *PointName* `With Coordinates` $Coordinates_1$ $Coordinates_2$ $Coordinates_3$

Parameter	Value	Default
PointName	`string`	`undefined`
Coordinates	`real_1 real_2 real_3`	`undefined`

Summary Defines a named point in space in terms of its coordinates.

9.5 Coupling BCs for Organic Material Decomposition

Aria provides the capability of coupling between porous (see section 5.11.2) and fluid blocks using flux-style boundary conditions. The boundary conditions described here are for a single region computation; for the most part, these boundary conditions can be used for a multi-region computation with the phrase 'one_region' omitted. These boundary conditions are imposed as Robin boundary conditions,

$$\begin{aligned} F_P &= \rho_F \mathbf{u}_F \cdot \mathbf{n} + \beta(p_g - p_F) \\ F_F &= \rho_g \mathbf{u}_g \cdot \mathbf{n} + \beta(p_F - p_g) \end{aligned} \tag{9.42}$$

where F_P is the flux into the porous region and F_F is the flux into the fluid region. The boundary conditions for the pressure mass balance equation on the porous side and the corresponding flux condition on the fluid side CVFEM continuity equation are written as

```
BC flux for mass_balance in gas_phase on surface = fluid_robin_coupled_one_region
BC flux for cvfem_continuity on surface = porous_robin_one_region
```

This boundary condition uses a penalty coefficient of the form

$$\beta = \frac{c_{scaling} \rho_g K}{\mu_g h} \tag{9.43}$$

where h is the mesh width adjacent to the interface (need to check what happens with a non-uniform mesh). The default value of $c_{scaling}$ is 0.001. If this parameter is too large, the increase in pressure might be underpredicted.

A distinguishing boundary condition is used on the fluid momentum equation,

$$u_j - (u^D_{j,n} + u^D_{j,t}) = 0, \tag{9.44}$$

where $u^D_{j,n}$ is the imposed normal component of velocity and $u^D_{j,t}$ is the imposed tangential component of velocity. The normal component is computed directly from the continuity flux at the interface,

$$u^D_{j,n} = \frac{F_F}{\rho_F} n_j. \tag{9.45}$$

The tangential component is based on a variation of the classical Beavers-Joseph-Saffman condition [21, 22] for the slip velocity which has been extended to non-planar surfaces in multidimensional flow [23], which defines a provisional model velocity

$$u^{\mathrm{BJS}}_j = -\frac{\sqrt{\bar{K}}}{\alpha \mu} (n_i \tau_{ij}) \tag{9.46}$$

where $\bar{K}$ is the permeability of the porous region at the interface, μ is the viscosity of the local fluid at the interface, τ_{ij} is the viscous stress tensor of the fluid at the interface, and α is a dimensionless model parameter that is a function of the microstructure of the porous material, which has been found to have typical values near 0.1 [22]. The tangential component of this vector quantity is used as the tangential component of the distinguishing condition velocity, and is computed as

$$u_{j,t}^{D} = u_j^{\mathrm{BJS}} - \left(u_k^{\mathrm{BJS}} n_k\right) n_j. \tag{9.47}$$

which is written in the input file as

```
BC Disting for cvfem_momentum on surface = porous_robin_one_region
```

The flux of a species k across a porous-fluid interface is

$$\mathbf{J}_P^{Y_k} = \mathbf{J}_F^{diff} + F_P \rho_g Y_{k,PF} + \left(\frac{\bar{D_k \rho_f}}{h}\right)(Y_{k,P} - Y_{k,F}) \tag{9.48}$$

where P refers to porous side, F refers to the fluid side and $Y_{k,PF}$ is the upwinded interface mass fraction, equivalent to h_{PF} from the enthalpy coupling. The mass fraction coupling boundary condition on the porous side is written as

```
BC flux for mass_balance in phase on surface = fluid_robin_coupled_one_region
```

The enthalpy coupling boundary conditions in the porous region are

```
BC flux for porous_enthalpy in phase on surface = fluid_robin_coupled_with_solid_convection_one_region
BC flux for porous_enthalpy in phase on surface = fluid_solid_convection_coupled_one_region
```

9.6 BC EDGE

Edge boundary conditions are implemented in Aria to introduce physics at the intersection between two surfaces. In 2D this is a point (the intersection of two element edges) and in 3D this is an edge (the intersection of two element faces). These edge conditions can be applied on any equation system.

9.6.1 BC EDGE = WETTING

Parameters `theta_s = REAL`

Example `BC Edge for momentum where surface_1 intersects surface_2 = wetting theta_s = 120.`

Description This condition applies the edge BC on the momentum equation for a wetting line defined as the intersection between surface_1 and surface_2. The parameter theta_s is the static angle between the two surfaces and must be known *a priori*. This edge BC applies a constant surface traction proportional to the surface tension force defined on surface_2 in the direction of the wetting angle supplied as theta_s. Note that the surface tension force must be supplied in a material block and applied as a parameter for surface_2 (see section 4.50).

This force is defined formally as:

$$\vec{f} = \gamma \hat{\tau}_i \tag{9.49}$$

where γ represents the surface tension in (N/m) and $\hat{\tau}_i$ is the unit vector defined as

$$\hat{\tau}_i = \hat{\tau}_w \cos\theta_s + \hat{n}_w \sin\theta_s \tag{9.50}$$

In the above equation, $\hat{\tau}_w$ and $\hat{n}_w$ are defined as the wall-tangent and wall-normal vectors of surface_1, repsectively and θ_s is the wetting angle paramater theta_s.

Chapter 10

Distinguishing Conditions

10.1 An Introduction to Distinguishing Conditions

Aria's *distinguishing condition* (DC) feature is an essential ingredient in solving many coupled physics problems. A distinguishing condition is really just another equation specification except that it typically replaces a regular equation on a subset of the domain such as a surface.

For example, in solving fluids problems with a free surface where the mesh boundary moves with the material, e.g., an ALE simulation, a kinematic condition is used to tie the mesh to the fluid on the free boundary. In this example, one of the mesh coordinates, say `MESH_DISPLACEMENTS_X`, is unknown. So, the equation that is normally used to solve for that component (the $x-$component of the `MESH` equation) is replaced with the kinematic condition: $\boldsymbol{n} \cdot \left(\boldsymbol{v} - \dot{\boldsymbol{d}} \right) - v_\circ = 0$.

An additional feature of distinguishing conditions is that multiple DCs for a given degree of freedom on a given surface are added together. This additive feature allows users to build up their own conditions from primitive ones.

An important thing to know about these conditions is that they are satisfied *weakly*. That is, the DC is multiplied by a finite element weight function and integrated over the surface. Consequently, the condition is only satisfied weakly and to within the tolerance of the nonlinear solver.

The remainder of this chapter contains a description of the primitive DCs that are available in Aria. Using the user plugin feature described in chapter 18 users can add their own, more complicated or specialized conditions. Input specification of distinguishing conditions on portions of the FEM mesh can be simplified by aggregation of mesh parts into Mesh Groups 6.26.

10.2 BC DISTING

BC DISTING for ***EQNAME***[_<Subindex>][_<Phase>]
ON ***MESH_PART*** = ***MODEL***[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description — Replaces the equations for ***EQNAME*** on ***MESH_PART*** with a distinguishing condition implemented by ***MODEL*** .

Summary — Prior to the assembly of the distinguishing conditions, the "normal" matrix and RHS entries are zeroed. So, these conditions do not rely on penalty parameters. Also, if multiple distinguishing conditions are supplied they are added together so the *sum* of the conditions is satisfied – that is, they are not individually satisfied. This allows you to construct complex expressions based on the available primitives and/or any plugins. See the `POLYNOMIAL` model below for an example combining two distinguishing conditions. Finally, it is worth noting

that these conditions are satisfied *weakly* and so they are only satisfied to within the tolerance of the nonlinear solver.

Parent Block(s) `ARIA REGION`

10.2.1 DARCY_LEAK

Description This model implements a vector distinguishing condition that allows fluid to "leak" through a boundary according to a Darcy's Law like relationship:

$$\boldsymbol{v} - \dot{\boldsymbol{x}} \cdot \boldsymbol{n}\boldsymbol{n} - \frac{K}{L\mu}\left(P - P_{ref}\right)\boldsymbol{n} = \boldsymbol{0}. \tag{10.1}$$

the velocity, $\dot{\boldsymbol{x}}$ is the time derivative of the mesh boundary position and P is the fluid pressure. K is the permeability of the boundary, μ is the viscosity, P_{ref} is a reference pressure (defaults to zero) and L is a length scale for the pressure drop (defaults to unity).

Note that the second term in 10.1 ensures that the resulting velocity is relative to any boundary motion that arises in moving mesh problems. For problems on fixed meshes this term is automatically excluded.

An alternate pressure field can be used by provided the optional parameter `PRESSURE`.

Parameters
```
K = REAL
[P_REF = REAL]
[L = REAL]
[PRESSURE = STRING]
```

Example
```
BC Disting For Momentum_A On surface_2 = Darcy_Leak K=1
BC Disting For Momentum_B On surface_2 = Darcy_Leak K=1e-7
```

10.2.2 KINEMATIC

Description This model implements the kinematic condition of the form

$$\boldsymbol{n} \cdot (\boldsymbol{v} - \dot{\boldsymbol{x}}) - v_\circ = 0 \tag{10.2}$$

where $\boldsymbol{n}$ is the outward unit normal to the boundary, $\boldsymbol{v}$ is the velocity, $\dot{\boldsymbol{x}}$ is the time derivative of the mesh boundary position and $v_\circ$ is the mass flux per unit mass across the interface, viz. a "leak" velocity. The leak velocity can be supplied by the `V0` parameter which defaults to zero.

Parameters
```
V0 = REAL
```

Example
```
BC Disting For Mesh_X On surface_2 = Kinematic v0=0
```

10.2.3 POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
BC Disting For Mesh_Y On surface_2 = Polynomial Variable=Time Order=1 C1=0.5

BC Disting For Mesh_Y On surface_2 = Polynomial Variable=Mesh_Displacement_Y
Order=1 C1=-1
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$\sum_{i=0}^{N} C_i X^i = 0 \tag{10.3}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

In the example given above, two distinguishing conditions are combined to give a composite function. In that example, the resulting conditions is

$$\frac{1}{2}t - \boldsymbol{d}_y = 0 \tag{10.4}$$

or

$$\boldsymbol{d}_y = \frac{1}{2}t \tag{10.5}$$

10.2.4 WETTING_SPEED_BLAKE_LS

Parameters

```
V_W = REAL
G = REAL
THETA = REAL
WIDTH = REAL
[TAU = REAL]
```

Example

```
BC Disting for Momentum_A on surface_3 = Wetting_Speed_Blake_LS V_w=1e-1 g=1
Width=1 Theta=60
```

Description

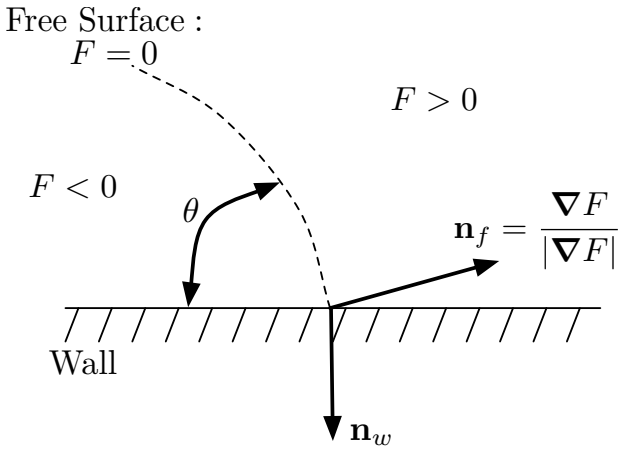
$$\boldsymbol{v} - \delta(F) v_w \sinh\left(g\left(\cos\theta_s - \cos\theta\right)\right)\boldsymbol{t}_w + \delta(F)\tau\dot{\boldsymbol{v}} = 0 \tag{10.6}$$

where $\boldsymbol{v}$ is the fluid velocity and $\boldsymbol{t}_w$ is the tangent to the wall. The function $\delta(F)$, where F is the level set distance field, is given by

$$\delta(F) = \frac{1}{2}\left(1 + \cos\frac{\pi F}{\alpha}\right) \tag{10.7}$$

when $|F| < \alpha$ and zero elsewhere. Here, α is the half of the `WIDTH` parameter. The term involving τ is a transient relaxation term. By default, $\tau = 0$.

This distinguishing condition is a function of the static and observed contact angles. The *static* contact angle θ_s, supplied by the `THETA` parameter, is fixed. The *observed* contact angle θ is computed from the current state of the solution as illustrated in the following diagram. The important point here is that the contact angle is measured through the *negative* side of the distance function which is denoted `PHASE_A` in Aria.



10.2.5 ELECTROOSMOTIC_VELOCITY

Parameters `C = REAL`

Example
```
BC Disting for Momentum on surface_3 = Electroosmotic_Velocity C=0.3
```

Description This distinguishing condition sets the velocity to simply be a constant multiple of the electric field,

$$\boldsymbol{v} - c\boldsymbol{E} = \boldsymbol{0} \tag{10.8}$$

where $\boldsymbol{v}$ is the fluid velocity, $\boldsymbol{E}$ is the electric field and c is the constant `C` supplied by the user.

Chapter 11

Volumetric Sources

This chapter documents the Source line commands within the current version of Aria. The Source line defines a model for a volumetric source term for a given equation 6. It is important to note that with the exception of sources defined in Volume Source command blocks for thermal problems 33.1 the source term must be explicitly defined with a Source command line. For both the single line and command block syntax, input specification of volumetric sources on portions of the FEM mesh can be simplified by aggregation of element blocks into Mesh Groups 6.26.

Source command lines appear in the Region scope of the input file as illustrated below.

```
.
.
Begin Procedure My_Aria_Procedure
   .
   .
   Begin Aria Region My_Region
    .
    .
    EQ Energy for Temperature on block_2 with Q1 using DIFF SRC
    Source for energy on block_2 = constant value = 0.2
    .
    .
   End

End
.
.
```

11.1 POINT SOURCE FOR ...

POINT SOURCE FOR *EQNAME* ON *MESH_PART* VALUE = Q X = x [Y = y [Z = z]]

Description — Arbitrary point source contributions.

Summary — This adds a point source with value `Q` at the specified position. Currently limited to constant and scalar sources. Only supported by the voltage equation (though extending it to other equations is simple, just ask). This line command syntax needs to change since the `ON` *MESH_PART* piece doesn't really make sense since you already supply the coordinates. You have to supply as many coordinate positions as the problem has dimensions.

Mathematically, the point source is represented as

$$q = Q\delta(\boldsymbol{x} - \boldsymbol{X}) \tag{11.1}$$

where $\delta()$ is the Dirac delta function which is zero everywhere except at the point $\boldsymbol{x} = \boldsymbol{X}$ where $\boldsymbol{x}$ is the physical coordinate and $\boldsymbol{X}$ is the position provided via the input. In finite elements, this source is integrated

$$\int_{\mathrm{V}} Q\delta(\boldsymbol{x} - \boldsymbol{X})\phi^i \,\mathrm{dV} = Q\phi^i(\boldsymbol{X}) \in \mathrm{Elem}_{\boldsymbol{X}}. \tag{11.2}$$

Here we employ the local support of the basis functions ϕ so that this is only evaluated in the elements containing the point $\boldsymbol{X}$, $\mathrm{Elem}_{\boldsymbol{X}}$.

Parent Block(s) `ARIA_REGION`

Example
```
POINT SOURCE FOR Voltage ON block_1 Value = 1 X = 0 Y = 0 Z = 0
```

11.2 VECTOR POINT SOURCE FOR ...

VECTOR POINT SOURCE FOR *EQNAME* ON *MESH_PART* V_X=*REAL* V_Y=*REAL* V_Z=*REAL* at X=*REAL* Y=*REAL* Z=*REAL*

Description Arbitrary vector point source contribution.

Summary This adds a vector point source with value (v_x, v_y, v_z) at the specified position. Currently limited to a constant value.

Mathematically, the point source is represented as

$$\boldsymbol{q} = \boldsymbol{Q}\delta(\boldsymbol{x} - \boldsymbol{X}) \tag{11.3}$$

where $\delta()$ is the Dirac delta function which is zero everywhere except at the point $\boldsymbol{x} = \boldsymbol{X}$ where $\boldsymbol{x}$ is the physical coordinate and $\boldsymbol{X}$ is the position provided via the input. In finite elements, this source is integrated

$$\int_{\mathrm{V}} \boldsymbol{Q}\delta(\boldsymbol{x} - \boldsymbol{X})\phi^i \,\mathrm{dV} = \boldsymbol{Q}\phi^i(\boldsymbol{X}) \in \mathrm{Elem}_{\boldsymbol{X}}. \tag{11.4}$$

Here we employ the local support of the basis functions ϕ so that this is only evaluated in the elements containing the point $\boldsymbol{X}$, $\mathrm{Elem}_{\boldsymbol{X}}$.

Parent Block(s) `ARIA_REGION`

Example
```
VECTOR POINT SOURCE FOR Momentum ON block_1 V_X=1 V_Y=1 V_Z=1 at X=0 Y=0
Z=0
```

11.3 SOURCE FOR ENERGY

SOURCE FOR ENERGY ON *MESH_PART* = *MODEL* [$\mathrm{param}_1 = \mathrm{val}_1$, $\mathrm{param}_2 = \mathrm{val}_2 \ldots$]

Description Arbitrary source contributions for the energy equation.

Summary Adds the source provided by *MODEL* to the energy equation.

Parent Block(s) ARIA_REGION

11.3.1 SOURCE FOR ENERGY = CALORE_USER_SUB

Parameters
```
NAME = STRING
TYPE = STRING
[MULTIPLIER = REAL]
[NR = INT]
[R0 = INT]
[R1 = INT(etc.)]
[NI = INT]
[I0 = INT]
[I1 = INT(etc.)]
```

Example
```
Source for Energy on block_10 = Calore_User_Sub name=w80afftuser
type=element NR=2 R0=1000 R1=0.7
```

Description NAME is the name of the user subroutine was registered with. NR and NI are the numbers of real and integer parameters, respectively. Each real (int) parameter is supplied by the Rn (In) parameter. **Note:** the parameters use zero based counting. The TYPE parameter denotes the API the user subroutine uses; currently Aria only supports types ELEMENT and NODE.

The optional MULTIPLIER parameter (defaults to 1.0) can be supplied to scale the output of the user subroutine, i.e., the flux f will be $f = mf_{user}$ where m is the multiplier and f_{user} is the output of the user subroutine.

See section 9.2.2 for a more complete example of a Calore user subroutine.

11.3.2 SOURCE FOR ENERGY = CHEMEQ_HEATING

Parameters
```
MODEL = STRING
[ POWER_OUTPUT = STRING]
```

Example
```
Source for Energy on block_10 = CHEMEQ_Heating model = reaction_model
power_output = VAR_NAME
```

Description Defines a heating source caused by chemical reaction. Here MODEL pertains to the paired CHEMEQ MODEL and CHEMEQ SOLVER command blocks that define the chemical system and how it can evolve. Chemical source contributions to the energy equation are further controlled through parameters in the CHEMEQ SOLVER command block.

When the POWER_OUTPUT option is present the energy source will be integrated over the associated element blocks and saved to the user specified global variable, VAR_NAME.

11.3.3 SOURCE FOR ENERGY = PID_CONTROLLED

Parameters

```
Max_Output = REAL
Setpoint_Function_Name = STRING
Control_Variable = STRING
[P = REAL]
[I = REAL]
[D = REAL]
[Band = REAL]
[Filter_Tau = REAL]
```

Example

```
Source for Energy on block_10 = PID_Controlled P=0.1 I=1e-4 Max_Output=20000
Setpoint_Function_Name=pid_setpoints1 Control_Variable=ctrlT1
```

Description

This source term imposes an energy flux using a PID controller. The controller setpoint vs. time is defined in a user function, and the controller feedback point uses a global variable (a data probe or Encore function).

The derivative term uses a filtered error value, using a low pass filter whose time constant is specified by the user.

Like a heater, this is only capable of providing a positive influx. Even if the controller output is negative, the applied heat flux will be truncated at 0. Likewise, the applied heat flux cannot exceed the specified max flux.

The nominal scaled PID controller output (between 0 and 1) is

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{\partial e_f}{\partial t} \tag{11.5}$$

where it is important to note that the K_p coefficient does not scale the others unless you have set it using the Band argument. In some PID implementations, the K_i and K_d coefficients are internally multiplied by K_p so care must be taken to provide the correct coefficients. This output $u(t)$ which is from 0 to 1 is scaled by the specified Max_Output value to produce the applied energy flux. This means you should not include the output magnitude scaling in your P, I, or D parameters.

The P, I, and D coefficients can be specified in a number of ways:

- Just give a 'Band' argument. This calculates P = 1/Band and uses the defaults of I = 0.01*P and D = 0.
- Give a 'Band' argument and non-defaults for I and/or D. This still multiplies whatever you give for I and D by P.
- Give a 'P' argument instead of 'Band'. NOTE: in this mode the I and D values are not scaled by P.

If you provide both P and Band, P is used, the Band argument is ignored, and I and D are not scaled.

The default values for I and D are 0.01 and 0 if not provided.

The setpoint, controller output, error, filtered error, and filtered error time derivative are all output to automatically created global variables for diagnostics and postprocessing.

11.3.4 SOURCE FOR ENERGY = PID_CONTROLLED_COOLER

Parameters
```
Max_Cooling_Power = REAL
Setpoint_Function_Name = STRING
Control_Variable = STRING
[P = REAL]
[I = REAL]
[D = REAL]
[Band = REAL]
[Filter_Tau = REAL]
```

Example
```
Source for Energy on block_1 = PID_Controlled_Cooler P=0.1 I=1e-4
Max_Cooling_Power=20000 Setpoint_Function_Name=pid_setpoints1
Control_Variable=ctrlT1
```

Description This is similar to PID_Controlled, however, the flux is applied to cool the volume instead of heat it. Max_Cooling_Power should be a positive value.

11.3.5 SOURCE FOR ENERGY = PID_CONTROLLED_BIDIRECTIONAL

Parameters
```
Max_Cooling_Power = REAL
Max_Heating_Power = REAL
Setpoint_Function_Name = STRING
Control_Variable = STRING
[P = REAL]
[I = REAL]
[D = REAL]
[Band = REAL]
[Filter_Tau = REAL]
```

Example
```
Source for Energy on block_1 = PID_Controlled_Bidirectional
P=0.1 I=1e-4 Max_Cooling_Power=20000 Max_Heating_Power=20000
Setpoint_Function_Name=pid_setpoints1 Control_Variable=ctrlT1
```

Description This is similar to PID_Controlled, however, the flux is applied to cool and heat the volume instead of just heating it. Max_Cooling_Power and Max_Heating_Power should both be positive values.

11.3.6 SOURCE FOR ENERGY = MELTING

Parameters
```
Ts = REAL
Tl = REAL
[ latent_heat = REAL]
```

Example
```
Source for Energy on block_1 = Melting Ts = 400 Tl = 410 latent_heat = 1e5
Source for Energy on block_1 = Melting Ts = 700 Tl = 725 latent_heat = 1e8
Source for Energy on block_1 = Melting Ts = 900 Tl = 950
```

Description Defines a heating source caused by a phase change. The phase change is modeled using a normal distribution spread over a temperature range defined by Ts and Tl. The distribution is sized so that 99% of the energy release occurs between Ts and Tl. The latent heat associated with the phase change can either be specified in-line with the optional "latent_heat" argument or by specifying the latent heat expression in the material model. If both are provided, the value specified in-line is used. The latent heat should be specified units of energy per mass (e.g. J/kg).

This model is preferable to the "Specific Heat = Use_Phase_Change" model because it does not limit the time step or create a discontinuous specific heat. It is also guaranteed to correctly capture the total energy source or sink regardless of the time step size.

Multiple phase transitions can be specified by simply adding multiple melting sources with different temperatures and latent heats.

11.3.7 SOURCE FOR ENERGY = COMPRESSIVE_WORK

Parameters `[MULTIPLIER = REAL]`

Example `Source For Energy on block_1 = Compressive_Work`

Description This source accounts for the heat generation or consumption due to compression:

$$q = -m\,p : \boldsymbol{\nabla \cdot v} \tag{11.6}$$

where p is the pressure, $\boldsymbol{\nabla \cdot v}$ is the divergence of the velocity field and m is a multiplier (`MULTIPLIER`) that defaults to 1.

11.3.8 SOURCE FOR ENERGY = CONSTANT

Parameters `VALUE = REAL`

Example `SOURCE FOR Energy ON block_1 = Constant Value=1041.3`

Description Supplies a constant energy source

11.3.9 SOURCE FOR ENERGY = CURING_FOAM_HEAT_OF_RXN

Parameters
```
VFRAC_SUBINDEX = INT
EXTENT_SUBINDEX = INT
H_RXN = REAL
```

Example
```
Source For Energy on block_1 = Curing_Foam_Heat_of_Rxn Vfrac_Subindex=1
Extent_Subindex=2 H_rxn=250
```

Description This source accounts for the heat of reaction of a curing epoxy foam, specifically:

$$q = \rho\,(1-\phi)\,\Delta H_{rxn}\frac{\partial \xi}{\partial t} \tag{11.7}$$

where ρ is the density of the fluid, ϕ is the volume fraction, ΔH_{rxn} is the heat of reaction and ξ is the extent of reaction.

NOTE: The volume fraction is assumed to be a `SPECIES` field with the subindex provided by the `VFRAC_SUBINDEX` parameter. Likewise, the extent of reaction field is assumed to be a `SPECIES` field with the subindex provided by the `EXTENT_SUBINDEX` parameter.

11.3.10 SOURCE FOR ENERGY = CURING_FOAM_LATENT_HEAT

Parameters

```
VFRAC_SUBINDEX = INT
H_EVAP = REAL
```

Example

```
Source For Energy on block_1 = Curing_Foam_Latent_Heat Vfrac_Subindex=1
H_evap=15
```

Description

This source accounts for the loss of energy due to evaporation of a curing epoxy foam, specifically:

$$q = \rho H_{evap} \frac{\partial \phi}{\partial t} \tag{11.8}$$

where ρ is the density of the fluid, H_{evap} is the latent heat of evaporation and ϕ is the volume fraction,

NOTE: The volume fraction is assumed to be a `SPECIES` field with the subindex provided by the `VFRAC_SUBINDEX` parameter.

11.3.11 SOURCE FOR ENERGY = CURING_FOAM_SPECIFIC_HEAT

Parameters

```
VFRAC_SUBINDEX = INT
[CP_FG = REAL]
[CP_E = REAL]
[PHI_ZERO = REAL]
```

Example

```
Source For Energy on block_1 = Curing_Foam_Specific_Heat Vfrac_Subindex=1
Cp_fG=1 Cp_e=1 phi_zero=0.2
```

Description

This source accounts for the loss of energy due to the variable specific heat for the special case where the specific heat material model is `CURING_FOAM`. See 4.48.6. Specifically, this source term is

$$q = -\rho T \left(\frac{\partial C_p}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} C_p \right) \tag{11.9}$$

$$= -\rho T b \left(\frac{\partial \phi}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \phi \right) \tag{11.10}$$

where ρ is the density of the fluid, T is the temperature, $\boldsymbol{v}$ is the velocity, ϕ is the volume fraction and b is as defined in `CURING_FOAM` specific heat material model (see 4.48.6).

NOTE: The volume fraction is assumed to be a `SPECIES` field with the subindex provided by the `VFRAC_SUBINDEX` parameter.

11.3.12 SOURCE FOR ENERGY = ENCORE_FUNCTION

Parameters
```
NAME = STRING
[EVAL_TYPE = STRING]
```

Example
```
SOURCE FOR Energy ON block_1 = Encore_Function Name=My_Function
```

Description

Using the Encore library, you can define your own functions in a number of ways, such as analytically in the input file, based on the values of a Sierra field or your own compiled functions. See the Encore documentation for more details.

As an example, this snippet of input can be added to the domain level of your input file:

```
Begin String Function My_Function
   Value is "200 + 1.0*t"
End
```

Then, this function can be used as shown in the example above.

The optional argument `EVAL_TYPE` can be used to select the particular function on the Encore function object. Valid options include `VALUE`, `DOT`, `GRADIENT`, `FLUX` and `STRESS`. If this option is not supplied Aria will make a sensible choice depending on the resulting expressions – `VALUE` for `NO_OP` expressions, `DOT` for `DT_OP` expressions and `GRADIENT` for `GRAD_OP` expressions.

11.3.13 SOURCE FOR ENERGY = JOULE_HEATING

Parameters
```
[VOLTAGE_SUBINDEX = INT]
```

Example
```
SOURCE FOR Energy ON block_1 = Joule_Heating
```

Description

This source term adds the volumetric heat source due to Joule heating, a.k.a., Ohmic heating or resistance heating.

$$H_V = J \cdot E \tag{11.11}$$

where J is the current density vector and E is the electric field vector.

When thermoelectric effects are negligible volumetric heating is effectively given by (see, Section 5.3)

$$\begin{aligned} H_V &= J^2 R && (11.12)\\ &= (-\sigma_e \boldsymbol{\nabla} V) \cdot (-\sigma_e \boldsymbol{\nabla} V)\, R && (11.13)\\ &= (-\sigma_e \boldsymbol{\nabla} V) \cdot (-\sigma_e \boldsymbol{\nabla} V)\, \frac{1}{\sigma_e} && (11.14)\\ &= \sigma_e\, (\boldsymbol{\nabla} V \cdot \boldsymbol{\nabla} V) && (11.15) \end{aligned}$$

where J is the current density, R is the resistivity, $\sigma_e = 1/R$ is the electrical conductivity and V is the voltage.

If thermoelectric effects are significant then a thermoelectric current density 4.6.4 should be used instead

$$J = -\sigma_e \left(\boldsymbol{\nabla} V + \alpha \boldsymbol{\nabla} T \right) \tag{11.16}$$

where α is the Seebeck coefficient and T is the temperature.

11.3.14 SOURCE FOR ENERGY = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[VARIABLE_OFFSET = REAL]
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
SOURCE For Energy on block_1= Polynomial Variable=Temperature Order=1
C0=401.0 C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$H_V = \sum_{i=0}^{N} C_i \left(X + X_\circ \right)^i \tag{11.17}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter, $X_\circ$ is an optional offset (`VARIABLE_OFFSET`, default is zero) and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

11.3.15 SOURCE FOR ENERGY = TBC_JOULE_HEATING

Parameters

```
Ua = REAL
Ub = REAL
V_NS = INT
Ti = REAL
CURRENT_LOAD = REAL
[VOLTAGE_SUBINDEX = INT]
```

Example

```
SOURCE FOR Energy ON block_1 = TBC_Joule_Heating Ua=1.4251 Ub=0.0004785
V_NS=1 Ti=298 CURRENT_LOAD=0.0017
```

Description

This source term adds the volumetric heat source due to Joule heating in a thermal battery cell volumetric heating is given by (see, Section 5.3)

$$H_{\mathrm{V}} = \left(U_\circ - V - T_i \frac{\partial U_\circ}{\partial T} \right) I_\circ \tag{11.18}$$

Here, $U_\circ$ is the open circuit potential which is given as a linear function in temperature T and U_a and U_b are the coefficients of that function. V is the cell potential which is taken as the voltage at the node given in the single-node nodeset number `V_NS` and $I_\circ$ is cell load.

For more details, see [24].

11.3.16 SOURCE FOR ENERGY = VISCOUS_DISSIPATION

Parameters `[MULTIPLIER = REAL]`

Example `Source For Energy on block_1 = Viscous_Dissipation`

Description This source accounts for the heat generation due to viscous dissipation:

$$q = m\,\boldsymbol{\tau} : \boldsymbol{\nabla v} \tag{11.19}$$

where $\boldsymbol{\tau}$ is the viscous stress tensor, $\boldsymbol{\nabla v}$ is the gradient of the velocity and m is a multiplier (`MULTIPLIER`) that defaults to 1. The exact form of $\boldsymbol{\tau}$ will depend on the `MOMENTUM_STRESS` model choice(s) for the material.

11.3.17 SOURCE FOR ENERGY = USER_FUNCTION

Parameters
```
NAME = STRING
X = STRING
```

Example
```
SOURCE For Energy on block_1 = User_Function Name=Function_Name
X=Independent_Variable
```

Description Source magnitude is defined by a tabular function of the independent variable `X`. The user specified tabular function `NAME` as a function of `X`scalar solution variable or time. The `X` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `X` argument is described in section 2.7.1. Although described here for the energy equation, this source term may be defined on other scalar equations.

11.3.18 SOURCE FOR ENERGY = ELECTRODE_OBJECT

See 11.5.3 for complete description.

11.4 SOURCE FOR MOMENTUM

SOURCE FOR MOMENTUM ON *MESH_PART* = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Arbitrary source contributions for the momentum equation.

Summary Adds the source provided by *MODEL* to the momentum equation.

Parent Block(s) `ARIA_REGION`

11.4.1 SOURCE FOR MOMENTUM = CONSTANT_VECTOR

Parameters
```
[X = REAL]
[Y = REAL]
[Z = REAL]
```

Example
```
SOURCE FOR momentum ON block_1 = CONSTANT_VECTOR Z=-9.8
```

Description This source term only applies to the momentum equation though it is automatically applied to the continuity equation in PSPG formulations.

`X`, `Y`, `Z` are the components of the vector source. This vector source is *not* multiplied by the density.

11.4.2 SOURCE FOR MOMENTUM = HYDROSTATIC

Parameters
```
GX = REAL
GY = REAL
GZ = REAL
[REF_DENSITY = REAL]
```

Example
```
SOURCE FOR momentum ON block_1 = HYDROSTATIC gx = 0 gy = 0 gz = -980
```

Description This source term only applies to the momentum equation though it is automatically applied to the continuity equation in PSPG formulations.

`GX`, `GY`, `GZ` are the components of the gravity vector $\boldsymbol{g}$ and `REF_DENSITY` is an constant, uniform reference density $\rho_\circ$. With density ρ, the hydrostatic source is defined as $(\rho - \rho_\circ)\boldsymbol{g}$.

The default value of the reference density is 0.

11.4.3 SOURCE FOR MOMENTUM = POTENTIAL

Parameters
```
None
```

Example
```
SOURCE FOR momentum ON block_1 = POTENTIAL
```

Description This source term applies the gradient of the potential field, defined by the POTENTIAL equation, as a momentum source. It is most commonly used when the potential field is defined as the hydrostatic potential, and is used in place of the HYDROSTATIC model discussed above.

11.4.4 SOURCE FOR MOMENTUM = ROTATING_BODY_FORCE

Parameters
```
G = REAL
FREQUENCY = REAL
[PHASE_SHIFT = REAL]
[REF_DENSITY = REAL]
```

Example `Source for Momentum on block_1 = Rotating_Body_Force g=9.8 frequency=2.5 phase_shift=90`

Description This source term only applies to the momentum equation though it is automatically applied to the continuity equation in PSPG formulations.

`GX`, `GY`, `GZ` are the components of the gravity vector $\boldsymbol{g}$ and `REF_DENSITY` is an constant, uniform reference density $\rho_\circ$. With density ρ, the hydrostatic source is defined as $(\rho - \rho_\circ)\boldsymbol{g}$.

The default value of the reference density is 0.

11.4.5 SOURCE FOR MOMENTUM = BOUSSINESQ

Parameters
```
TEMP_REF = REAL
VOL_EXP = REAL
GX = REAL
GY = REAL
GZ = REAL
```

Example `SOURCE FOR momentum ON block_1 = BOUSSINESQ vol_exp=0.1 temp_ref=298.15 gx = 0 gy = 0 gz = -980`

Description This source term only applies to the momentum equation though it is automatically applied to the continuity equation in PSPG formulations.

`GX`, `GY`, `GZ` are the components of the gravity vector $\boldsymbol{g}$. `VOL_EXP` is the volume expansion coefficient α and `TEMP_REF` is the reference temperature T_{ref}. With density ρ and temperature T the Boussinesq source is defined as $\rho\boldsymbol{g}\alpha(T - T_{ref})$.

11.5 SOURCE FOR CURRENT

SOURCE FOR CURRENT ON *MESH_PART* = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Arbitrary source contributions for the current equation.

Summary Adds the source provided by *MODEL* to the current equation.

Parent Block(s) `ARIA_REGION`

11.5.1 SOURCE FOR CURRENT = BUTLER_VOLMER_SIMPLE

Parameters
```
A = REAL
C_A = REAL
C_C = REAL
U = REAL
Sign = INT
[V1_SUBINDEX = INT]
[V2_SUBINDEX = INT]
```

Example

```
SOURCE FOR CURRENT_1 ON block_1 = Butler_Volmer_Simple A=1.0 C_a=1.0
C_c=-1.0 U=0.2 Sign=-1
SOURCE FOR CURRENT_2 ON block_1 = Butler_Volmer_Simple A=1.0 C_a=1.0
C_c=-1.0 U=0.0 Sign=+1
```

Description

This model implements a very simple form of the Butler-Volmer reaction kinetics. It is intended for developmental and demonstrational purposes only.

This source term has the following form (see, also, equation 5.35)

$$R_{V,k} = A\left(e^{c_a(V_1-V_2-U)} - e^{-c_c(V_1-V_2-U)}\right) \tag{11.20}$$

where V_1 is is the first electric potential field and V_2 is the second electric potential field. By default V_1 is `VOLTAGE_1` and V_2 is `VOLTAGE_2` but the subindices may be changed using the optional `V1_SUBINDEX` and `V2_SUBINDEX` options.

11.5.2 SOURCE FOR CURRENT = POLYNOMIAL

Parameters

```
VARIABLE = STRING
ORDER = INT
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example

```
SOURCE For Current on block_1 = Polynomial Variable=Temperature Order=1
C0=401.0 C1=88.5
```

Description

Arbitrary order polynomial function of a specified scalar variable.

$$H_V = \sum_{i=0}^{N} C_i X^i \tag{11.21}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

11.5.3 SOURCE FOR CURRENT = ELECTRODE_OBJECT

Parameters

```
electrodeFile = STRING
[F = REAL]
[kmolConversion = REAL]
[meterConversion = REAL]
[JConversion = REAL]
```

Example

```
SOURCE FOR CURRENT in SOLID_PHASE ON block_1 = ELECTRODE_OBJECT F=96485
electrodeFile=cathode.inp kmolConversion=1000
```

Description This model uses the Cantera Electrode object to compute source terms for battery models. Generally this source term will need to be applied to the following equations:

- CURRENT in SOLID_PHASE
- CURRENT in LIQUID_PHASE
- POROUS_SPECIES of (SPECIES_NAME) in LIQUID_PHASE
- [ENERGY]

electrodeFile specifies the input file that will be used to initialize the Electrode objects.
F specifies the value of Faraday's constant in the unit system of the simulation.
kmolConversion specifies the conversion factor from kmol to the unit system of the simulation.
meterConversion specifies the conversion factor from meters to the unit system of the simulation.
JConversion specifies the conversion factor from Joules to the unit system of the simulation.

11.6 SOURCE FOR SPECIES

SOURCE FOR SPECIES ON *MESH_PART* = *MODEL* [$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Arbitrary source contributions for the species equation.

Summary Adds the source provided by *MODEL* to the species equation.

Parent Block(s) `ARIA_REGION`

11.6.1 SOURCE FOR SPECIES = CURING_FOAM_EXTENT

Parameters
```
K = REAL
E = REAL
R = REAL
N = REAL
```

Example
```
Source For Species_2 on block_1 = Curing_Foam_Extent k=1.145e5 E=10
R=8.314472E3 n=1.3
```

Description This source accounts for the reaction of a curing epoxy foam, specifically:

$$q = ke^{E/RT}\left(1-\xi\right)^n \tag{11.22}$$

where T is the temperature and ξ is the extent of reaction.

11.6.2 SOURCE FOR SPECIES = CURING_FOAM_VFRAC

Parameters
```
A = REAL
B = REAL
C = REAL
T_BOILING = REAL
```

Example `Source For Species_2 on block_1 = Curing_Foam_Vfrac a=1 b=0 c=2e-3 T_BOILING=473.15`

Description This source accounts for the change in volume fraction with temperature.

$$q = \left(a + bT + cT^2\right)_{\exp} \frac{\partial T}{\partial t} \tag{11.23}$$

where T is the temperature. This source term is only active when $T >= T_{boiling}$.

11.6.3 SOURCE FOR SPECIES = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example `Source For Species on block_1 = Polynomial Variable=Temperature Order=1 C0=401.0 C1=88.5`

Description Arbitrary order polynomial function of a specified scalar variable.

$$H_V = \sum_{i=0}^{N} C_i X^i \tag{11.24}$$

Here, N is the order of the polynomial provided by the `ORDER` parameter and X is the variable supplied by the `VARIABLE` parameter and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The `VARIABLE` argument can be `TIME` or any internal Expression that evaluates to a scalar. For the latter case, the format of the `VARIABLE` argument is described in section 2.7.1.

11.7 SOURCE FOR POROUS_SPECIES

SOURCE FOR POROUS_SPECIES ON *MESH_PART* = *MODEL*[param$_1$ = val$_1$, param$_2$ = val$_2$ ...]

Description Arbitrary source contributions for the porous_species equation.

Summary Adds the source provided by *MODEL* to the porous_species equation.

Parent Block(s) `ARIA_REGION`

11.7.1 SOURCE FOR POROUS_SPECIES = ELECTRODE_OBJECT

See 11.5.3 for complete description.

11.8 SOURCE FOR VOLTAGE

SOURCE FOR VOLTAGE ON *MESH_PART* = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Arbitrary source contributions for the voltage equation.

Summary Adds the source provided by *MODEL* to the voltage equation.

Parent Block(s) ARIA_REGION

11.8.1 SOURCE FOR VOLTAGE = POLYNOMIAL

Parameters
```
VARIABLE = STRING
ORDER = INT
[C0 = REAL]
[C1 = REAL]
...
[CN = REAL]
```

Example
```
Source For Voltage on block_1 = Polynomial Variable=Temperature Order=1
C0=401.0 C1=88.5
```

Description Arbitrary order polynomial function of a specified scalar variable.

$$H_V = \sum_{i=0}^{N} C_i X^i \tag{11.25}$$

Here, N is the order of the polynomial provided by the ORDER parameter and X is the variable supplied by the VARIABLE parameter and C_i are the supplied coefficients. Coefficients that are not supplied default to a value of zero. The VARIABLE argument can be TIME or any internal Expression that evaluates to a scalar. For the latter case, the format of the VARIABLE argument is described in section 2.7.1.

11.9 SOURCE FOR POTENTIAL

SOURCE FOR POTENTIAL ON *MESH_PART* = *MODEL*[$\text{param}_1 = \text{val}_1$, $\text{param}_2 = \text{val}_2 \ldots$]

Description Arbitrary source contributions for the potential equation.

Summary Adds the source provided by *MODEL* to the potential equation.

Parent Block(s) ARIA_REGION

11.9.1 SOURCE FOR POTENTIAL = HYDROSTATIC

Parameters

```
GX = REAL
GY = REAL
GZ = REAL
[REF_DENSITY = REAL]
```

Example

```
SOURCE FOR Potential ON block_1 = HYDROSTATIC gx = 0 gy = 0 gz = -980
```

Description

This source term only applies a hydrostatic potential source term to the POTENTIAL equation, defining the potential. It can then be applied to the momentum equation through a separate potential source term in the momentum equation.

`GX`, `GY`, `GZ` are the components of the gravity vector $\boldsymbol{g}$ and `REF_DENSITY` is an constant, uniform reference density $\rho_\circ$. With density ρ, the hydrostatic source is defined as $(\rho - \rho_\circ)\boldsymbol{g}$.

The default value of the reference density is 0.

Chapter 12

Constraint Conditions

At present Aria supports only a two types of constraints. However, many problem inputs can often be partially constrained using a PID controller.

12.0.2 Suspension Average Volume Fraction

12.1 CONSTRAIN

`CONSTRAIN AVERAGE_VOLUME_FRACTION phi =` C_0

Description Integral constraint for the average particle volume fraction in a SUSPENSION problem.

Summary Constrains ϕ throughout the problem domain to achieve the specified value of average volume particle fraction in accordance with the relation

$$C_0 = \left[\int_{\mathrm{V}_\phi} \mathrm{dV}\right]^{-1} \int_{\mathrm{V}_\phi} \phi \mathrm{dV} \tag{12.1}$$

over element blocks where the SUSPENSION equation is defined.

Parent Block(s) `ARIA_REGION`

12.1.1 Embedded Submodel

Beta Capability: This feature is not sufficiently tested to be classified as a full production capability.

The embedded submodel is a type of constraint that enables one to independently include a shell or bar feature in an existing contiguously meshed model. Here the shell or bar DOF are constrained/tied to the DOF evaluations of the contiguous model using Multi-Point Constraints.

12.2 Submodel

Scope: Equation System

```
Begin Submodel SubModel_name
    Clip Isoparametric Coords
    Embedded Blocks {= | are | is} VolumeList...
    Enclosing Blocks {= | are | is} VolumeList...
    Expand Box Percentage {= | are | is} Percentage
    Interaction Output {= | are | is} NonconformalOutput
    Search Method {= | are | is} SearchMethod
    Search Tolerance {= | are | is} Tolerance
End
```

Summary — Contains the commands needed to define an Aria submodel.

Description — This block command is used to define the embedded block submodel.

12.2.1 Clip Isoparametric Coords

Scope: Submodel

Summary — This command is no longer used. It remains available for backwards compatibility until version 4.50.

12.2.2 Embedded Blocks

Scope: Submodel

`Embedded Blocks {= | are | is} VolumeList...`

Parameter	Value	Default
VolumeList	`string...`	`undefined`

Summary — Specifies the element blocks to be embedded within another volume.

12.2.3 Enclosing Blocks

Scope: Submodel

`Enclosing Blocks {= | are | is} VolumeList...`

Parameter	Value	Default
VolumeList	`string...`	`undefined`

Summary — Specifies the element blocks in which EMBEDDED BLOCKS can reside.

12.2.4 Expand Box Percentage

Scope: Submodel

`Expand Box Percentage {= | are | is}` *Percentage*

Parameter	Value	Default
Percentage	`real`	`1.0`

Summary This command is no longer used. It remains available for backwards compatibility until version 4.50.

12.2.5 Interaction Output

Scope: Submodel

`Interaction Output {= | are | is}` *NonconformalOutput*

Parameter	Value	Default
NonconformalOutput	`{active	all	error	none	summary}`	`NONE`

Summary When this line is present, detailed information of the contact interactions are output.

12.2.6 Search Method

Scope: Submodel

`Search Method {= | are | is}` *SearchMethod*

Parameter	Value	Default
SearchMethod	`{boost	kdtree}`	`NONE`

Summary Set the search method to be used with nonconformal contact.

12.2.7 Search Tolerance

Scope: Submodel

`Search Tolerance {= | are | is}` *Tolerance*

Parameter	Value	Default
Tolerance	`real`	`1.0`

Summary Assign the normal search tolerance in the length units of the model. This will override the automatic tolerancing.

Chapter 13

Element Death

The element death capability is used to define conditions under which elements will be excluded from the normal set of calculations. This capability is implemented in two different flavors, one where elements are explicitly removed (Standard Element Death) and another (CDFEM) in which elements are subdivided in a manner consistent with the element death criterion. Generally speaking there is computational economy in the usage of standard element death over CVFEM death at the expense of resolution in front tracking. With a few exceptions both element death implementations utilize essentially the same command directives.

Usage of the death capability is comprised of two parts, definition of criterion and definition of scope. Each death criterion definition is must be named uniquely (DeathName) and its scope will be a specific set of mesh entities, surfaces or blocks of the input mesh. Element death is a dynamic process in which a variable (DEATH_STATUS) on existing elements is modified to denote whether an element is currently active, DEATH_STATUS=1 or inactive, DEATH_STATUS=0.

Visualization of a model with the "dead" elements removed is made possible by use of the DEATH_STATUS variable. Since DEATH_STATUS=0 denotes dead elements, the DEATH_STATUS variable can then be used in conjunction with any visualization thresholding utility to animate element death process.

During a simulation in which element death is active one can monitor the extent of element death through two values output to the log file, KILLED_ELEMENTS and TOTAL_DEAD_ELEMENTS. KILLED_ELEMENTS reports the number of elements eliminated after completion of the time step while TOTAL_DEAD_ELEMENTS is the cumulative number of elements deleted. Both KILLED_ELEMENTS and TOTAL_DEAD_ELEMENTS are also available for output to the ExodusII database as global variables under the same name (e.g. global variables = killed_elements).

Since the death criterion specification will normally involve an internally stored Field, one must use the complete name of that Field when defining element death. As an example, element death named criteria DEATH_TEMP8 based upon the average temperature solution on BLOCK_3 exceeding 322K might be defined as:

```
Begin Element Death death_temp8
  add volume block_3
  Criterion is Avg nodal value of solution->Temperature > 322.0
End
```

When element death is applied to elements at an exposed boundary an additional step is required to preserve the application of existing boundary conditions at newly exposed surfaces resulting from element removal. This is accomplished by associating the surface of the element block where element death is applied as part of the boundary condition surface. As an example, a flux boundary condition applied to a surface of the use case previously given would require the syntax:

```
Begin Heat Flux Boundary Condition X-face
  Add Surface surface_4
  Add Surface surface_death_temp8
  Flux = 100.
End
```

In most cases the element death criteria will be evaluated based upon values computed within Aria. Here the death criteria are evaluated at the end of a solution step. However, it is also possible to evaluate the criteria based upon externally supplied "transfer" variables. In this case one can force the death criteria to be evaluated at the beginning of time step using the TRANSFER ELEMENT DEATH command line. a solution step

13.1 Element Death

Scope: Aria Region

Begin Element Death *Death Name*

Add Volume *Block Name...*

Criterion {= | are | is} *DeathCriterionType Variable Name DeathCriterionCompareType Threshold Value*

End

Summary — Allows the specification of an element death criterion to one or more element volumes. The group of element volumes plus a death criterion make up a "death instance".

13.1.1 Add Volume

Scope: Element Death

Add Volume *Block Name...*

Parameter	Value	Default
Block Name	string...	undefined

Summary — Specifies the names identifying an element volume to which an element death criterion is to be applied. Line can be repeated as needed to add multiple volumes. A group of element volumes plus a death criterion make up a "death instance".

13.1.2 Criterion

Scope: Element Death

`Criterion {= | are | is}` *`DeathCriterionType Variable Name DeathCriterionCompareType Threshold Value`*

Parameter	Value	Default
DeathCriterionType	`{avg nodal value of | death criterion type undefined | element value of | max nodal value of | min nodal value of}`	`undefined`
Variable Name	`string`	`undefined`
DeathCriterionCompareType	`{< | <= | = | > | >= | death criterion compare type undefined}`	`undefined`
Threshold Value	`real`	`undefined`

Summary — The death criterion for one or more element volumes. Currently designed to support the specification of a threshold for a scalar variable defined on either a node or element. For nodal variables, various operations such as Min., Max. and Avg. are provided to reduce the nodal values to a single element value. For an element variable no operations are given as there is only one value per element. A group of element volumes plus a death criterion make up a "death instance". Currently only one criterion is allowed per death instance.

13.1.3 Transfer Element Death

Scope:

Summary — Cause execution of element death to occur at beginning of a time step rather than at the end of the step.

Chapter 14

Solution Control Reference

14.1 Overview

All Aria 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

   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. The details concerning time integration parameters are included in the section on Time Integration 16. 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

      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

         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
```

14.2 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.

14.2.1 Use System

Scope: Solution Control Description

```
Use System Name
```

Parameter	Value	Default
Name	string	undefined

Summary This set the name of which system to use.

14.3 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 commmand controls which one is to be used.

14.3.1 Adapt

Scope: System

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

Parameter	Value	Default
Region_name	string...	undefined
Field_name	string...	undefined

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

14.3.2 Compute Indicator On

Scope: System

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

Parameter	Value	Default
Region_name	string...	undefined
Indicator_name	string...	undefined

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

14.3.3 Event

Scope: System

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

Parameter	Value	Default
Name	`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.

14.3.4 Execute Postprocessor Group

Scope: System

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

Parameter	Value	Default
Group_name	`string...`	`undefined`
Region_name	`string...`	`undefined`

Summary Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

14.3.5 Indicatemarkadapt

Scope: System

`Indicatemarkadapt` *`Region_name`* `Using` *`Indicator Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	`string`	`undefined`
Indicator	`string`	`undefined`
Marker	`string`	`undefined`

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

14.3.6 Mark

Scope: System

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

Parameter	Value	Default
Region_name	`string...`	`undefined`
Marker_name	`string...`	`undefined`

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

14.3.7 Markadapt

Scope: System

`Markadapt` *`Region_name`* `Using` *`Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	string	undefined
Marker	string	undefined

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

14.3.8 Output

Scope: System

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

Parameter	Value	Default
Name	string	undefined

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

14.3.9 Simulation Max Global Iterations

Scope: System

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

Parameter	Value	Default
Number	integer	undefined

Summary The Total number of Solves.

14.3.10 Simulation Start Time

Scope: System

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

Parameter	Value	Default
Number	real	undefined

Summary Simulation starting time. (by default 0.0)

14.3.11 Simulation Termination Time

Scope: System

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

Parameter	Value	Default
Number	real	undefined

Summary The drop dead time.

14.3.12 Transfer

Scope: System

```
Transfer Name [ When When-expression ]
```

Parameter	Value	Default
Name	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.

14.3.13 Use Initialize

Scope: System

```
Use Initialize Name
```

Parameter	Value	Default
Name	string	undefined

Summary This set the name of which initialization to use.

14.4 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.

14.4.1 Adapt

Scope: Transient

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

Parameter	Value	Default
Region_name	string...	undefined
Field_name	string...	undefined

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

14.4.2 Advance

Scope: Transient

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

Parameter	Value	Default
Name	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.

14.4.3 Compute Indicator On

Scope: Transient

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

Parameter	Value	Default
Region_name	string...	undefined
Indicator_name	string...	undefined

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

14.4.4 Event

Scope: Transient

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

Parameter	Value	Default
Name	`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.

14.4.5 Execute Postprocessor Group

Scope: Transient

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

Parameter	Value	Default
Group_name	`string...`	`undefined`
Region_name	`string...`	`undefined`

Summary Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

14.4.6 Indicatemarkadapt

Scope: Transient

`Indicatemarkadapt` *`Region_name`* `Using` *`Indicator Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	`string`	`undefined`
Indicator	`string`	`undefined`
Marker	`string`	`undefined`

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

14.4.7 Involve

Scope: Transient

`Involve` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

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

14.4.8 Mark

Scope: Transient

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

Parameter	Value	Default
Region_name	`string...`	`undefined`
Marker_name	`string...`	`undefined`

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

14.4.9 Markadapt

Scope: Transient

`Markadapt` *`Region_name`* `Using` *`Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	`string`	`undefined`
Marker	`string`	`undefined`

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

14.4.10 Output

Scope: Transient

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

Parameter	Value	Default
Name	`string`	`undefined`

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

14.4.11 Transfer

Scope: Transient

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

Parameter	Value	Default
Name	`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.

14.5 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.

14.5.1 Adapt

Scope: Nonlinear

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

Parameter	Value	Default
Region_name	string...	undefined
Field_name	string...	undefined

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

14.5.2 Advance

Scope: Nonlinear

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

Parameter	Value	Default
Name	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.

14.5.3 Compute Indicator On

Scope: Nonlinear

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

Parameter	Value	Default
Region_name	string...	undefined
Indicator_name	string...	undefined

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

14.5.4 Event

Scope: Nonlinear

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

Parameter	Value	Default
Name	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.

14.5.5 Execute Postprocessor Group

Scope: Nonlinear

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

Parameter	Value	Default
Group_name	string...	undefined
Region_name	string...	undefined

Summary Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

14.5.6 Indicatemarkadapt

Scope: Nonlinear

`Indicatemarkadapt` *`Region_name`* `Using` *`Indicator Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	string	undefined
Indicator	string	undefined
Marker	string	undefined

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

14.5.7 Involve

Scope: Nonlinear

`Involve` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

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

14.5.8 Mark

Scope: Nonlinear

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

Parameter	Value	Default
Region_name	`string...`	`undefined`
Marker_name	`string...`	`undefined`

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

14.5.9 Markadapt

Scope: Nonlinear

`Markadapt` *`Region_name`* `Using` *`Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	`string`	`undefined`
Marker	`string`	`undefined`

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

14.5.10 Output

Scope: Nonlinear

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

Parameter	Value	Default
Name	`string`	`undefined`

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

14.5.11 Transfer

Scope: Nonlinear

```
Transfer Name [ When When-expression ]
```

Parameter	Value	Default
Name	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.

14.6 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.

14.6.1 Adapt

Scope: Subcycle

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

Parameter	Value	Default
Region_name	string...	undefined
Field_name	string...	undefined

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

14.6.2 Advance

Scope: Subcycle

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

Parameter	Value	Default
Name	`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.

14.6.3 Compute Indicator On

Scope: Subcycle

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

Parameter	Value	Default
Region_name	`string...`	`undefined`
Indicator_name	`string...`	`undefined`

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

14.6.4 Event

Scope: Subcycle

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

Parameter	Value	Default
Name	`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.

14.6.5 Execute Postprocessor Group

Scope: Subcycle

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

Parameter	Value	Default
Group_name	`string...`	`undefined`
Region_name	`string...`	`undefined`

Summary Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

14.6.6 Indicatemarkadapt

Scope: Subcycle

`Indicatemarkadapt` *`Region_name`* `Using` *`Indicator Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	string	undefined
Indicator	string	undefined
Marker	string	undefined

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

14.6.7 Involve

Scope: Subcycle

`Involve` *`Name`*

Parameter	Value	Default
Name	string	undefined

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

14.6.8 Mark

Scope: Subcycle

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

Parameter	Value	Default
Region_name	string...	undefined
Marker_name	string...	undefined

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

14.6.9 Markadapt

Scope: Subcycle

`Markadapt` *`Region_name`* `Using` *`Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	string	undefined
Marker	string	undefined

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

14.6.10 Output

Scope: Subcycle

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

Parameter	Value	Default
Name	string	undefined

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

14.6.11 Transfer

Scope: Subcycle

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

Parameter	Value	Default
Name	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.

14.7 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.

14.7.1 Adapt

Scope: Sequential

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

Parameter	Value	Default
Region_name	`string...`	`undefined`
Field_name	`string...`	`undefined`

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

14.7.2 Advance

Scope: Sequential

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

Parameter	Value	Default
Name	`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.

14.7.3 Compute Indicator On

Scope: Sequential

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

Parameter	Value	Default
Region_name	`string...`	`undefined`
Indicator_name	`string...`	`undefined`

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

14.7.4 Event

Scope: Sequential

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

Parameter	Value	Default
Name	`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.

14.7.5 Execute Postprocessor Group

Scope: Sequential

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

Parameter	Value	Default
Group_name	`string...`	`undefined`
Region_name	`string...`	`undefined`

Summary Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

14.7.6 Indicatemarkadapt

Scope: Sequential

`Indicatemarkadapt` *`Region_name`* `Using` *`Indicator Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	`string`	`undefined`
Indicator	`string`	`undefined`
Marker	`string`	`undefined`

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

14.7.7 Involve

Scope: Sequential

`Involve` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

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

14.7.8 Mark

Scope: Sequential

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

Parameter	Value	Default
Region_name	`string...`	`undefined`
Marker_name	`string...`	`undefined`

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

14.7.9 Markadapt

Scope: Sequential

`Markadapt` *`Region_name`* `Using` *`Marker`* `[ When` *`When-expression`* `]`

Parameter	Value	Default
Region_name	string	undefined
Marker	string	undefined

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

14.7.10 Output

Scope: Sequential

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

Parameter	Value	Default
Name	string	undefined

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

14.7.11 Transfer

Scope: Sequential

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

Parameter	Value	Default
Name	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.

14.8 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 commmand controls which one is to be used.

14.8.1 Advance

Scope: Initialize

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

Parameter	Value	Default
Name	`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.

14.8.2 Event

Scope: Initialize

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

Parameter	Value	Default
Name	`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.

14.8.3 Involve

Scope: Initialize

`Involve` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

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

14.8.4 Transfer

Scope: Initialize

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

Parameter	Value	Default
Name	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.

14.9 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.

14.9.1 Converged When

Scope: Parameters For

```
Converged When Convergence-expression
```

Parameter	Value	Default
Convergence-expression	(expression)	undefined

Summary Set the convergence expression.

14.9.2 Incremental Number Of Steps

Scope: Parameters For

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

Parameter	Value	Default
Number	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

14.9.3 Initial Deltat

Scope: Parameters For

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

Parameter	Value	Default
Number	real	undefined

Summary Assign an initial delta T

14.9.4 Number Of Adaptivity Steps

Scope: Parameters For

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

Parameter	Value	Default
Number	integer	undefined

Summary The number steps to run the time or nonlinear loop

14.9.5 Number Of Steps

Scope: Parameters For

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

Parameter	Value	Default
Number	integer	undefined

Summary The number steps to run the time for nonlinear loop

14.9.6 Reinitialize Transient

Scope: Parameters For

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

14.9.7 Start Time

Scope: Parameters For

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

Parameter	Value	Default
Number	`real`	`undefined`

Summary Assign a start time.

14.9.8 Termination Time

Scope: Parameters For

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

Parameter	Value	Default
Number	`real`	`undefined`

Summary Assign a final time to stop

14.9.9 Time Step Quantum

Scope: Parameters For

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

Parameter	Value	Default
TimeStepQuantum	`real`	`undefined`

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

14.9.10 Time Step Style

Scope: Parameters For

`Time Step Style` *`TimeStepStyle...`*

Parameter	Value	Default
TimeStepStyle	`{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.

14.9.11 Total Change In Time

Scope: Parameters For

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

Parameter	Value	Default
Number	`real`	`undefined`

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

14.9.12 Advance

Scope:

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

Parameter	Value	Default
Name	`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.

14.9.13 Converged When

Scope:

`Converged When` *`Convergence-expression`*

Parameter	Value	Default
Convergence-expression	`(expression)`	`undefined`

Summary Set the convergence expression.

14.9.14 Event

Scope:

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

Parameter	Value	Default
Name	`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.

14.9.15 Initial Deltat

Scope:

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

Parameter	Value	Default
Number	`real`	`undefined`

Summary Assign an initial delta T

14.9.16 Involve

Scope:

```
Involve Name
```

Parameter	Value	Default
Name	`string`	`undefined`

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

14.9.17 Number Of Adaptivity Steps

Scope:

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

Parameter	Value	Default
Number	`integer`	`undefined`

Summary The number steps to run the time or nonlinear loop

14.9.18 Number Of Steps

Scope:

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

Parameter	Value	Default
Number	`integer`	`undefined`

Summary The number steps to run the time for nonlinear loop

14.9.19 Output

Scope:

```
Output Name [ When When-expression ]
```

Parameter	Value	Default
Name	`string`	`undefined`

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

14.9.20 Reinitialize Transient

Scope:

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

14.9.21 Simulation Max Global Iterations

Scope:

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

Parameter	Value	Default
Number	`integer`	`undefined`

Summary The Total number of Solves.

14.9.22 Simulation Start Time

Scope:

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

Parameter	Value	Default
Number	`real`	`undefined`

Summary Simulation starting time. (by default 0.0)

14.9.23 Simulation Termination Time

Scope:

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

Parameter	Value	Default
Number	`real`	`undefined`

Summary The drop dead time.

14.9.24 Start Time

Scope:

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

Parameter	Value	Default
Number	`real`	`undefined`

Summary Assign a start time.

14.9.25 Termination Time

Scope:

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

Parameter	Value	Default
Number	`real`	`undefined`

Summary Assign a final time to stop

14.9.26 Time Step Quantum

Scope:

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

Parameter	Value	Default
TimeStepQuantum	`real`	`undefined`

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

14.9.27 Time Step Style

Scope:

`Time Step Style` *`TimeStepStyle...`*

Parameter	Value	Default
TimeStepStyle	`{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.

14.9.28 Total Change In Time

Scope:

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

Parameter	Value	Default
Number	`real`	`undefined`

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

14.9.29 Transfer

Scope:

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

Parameter	Value	Default
Name	`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.

14.9.30 Use Initialize

Scope:

`Use Initialize` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary This set the name of which initialization to use.

14.9.31 Use System

Scope:

`Use System` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary This set the name of which system to use.

Chapter 15

Transfer Reference

15.1 Overview

The Sierra Transfer utility provides the means by which to communicate data between two Sierra application Regions. The 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. Interpolate 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 15.1, 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 15.2. 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 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.

Both Aria solution Fields and User Fields 2.10 can be transferred to/from other Sierra application Regions. Transfer of data into Aria requires that provision be made for a Field storage on the Aria Region. Storage for solution Field (DOF) transfer into Aria is implicitly provided by the presence of an Aria equation command line with term type of XFER (Chapter 6). Storage for non-DOF Fields is implicitly provided through the presence of a User Field command line 2.11. These non-DOF Fields are internally used or modified by virtue of being referenced in other command lines such as material models, boundary conditions,

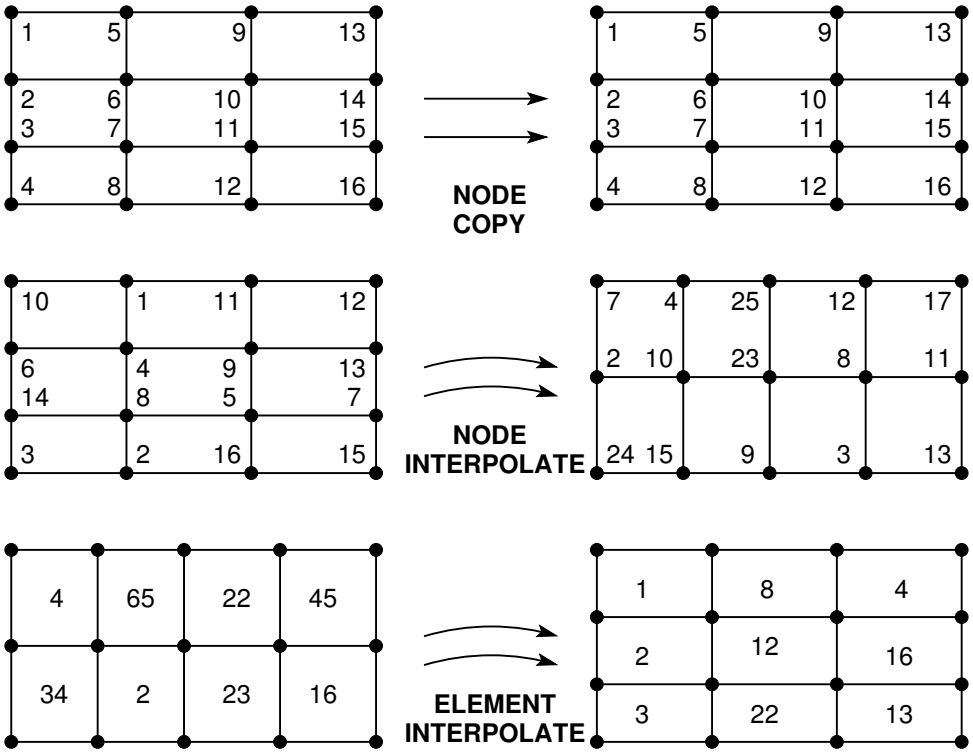


Figure 15.1. Valid Transfer Operations

initial conditions or source terms.

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
  .
  Begin Procedure My_Aria_Procedure
     .
    Begin Solution Control Description
       Use System Main
       Begin System Main
          Begin Sequential MySolveBlock
             Advance first_Region
```

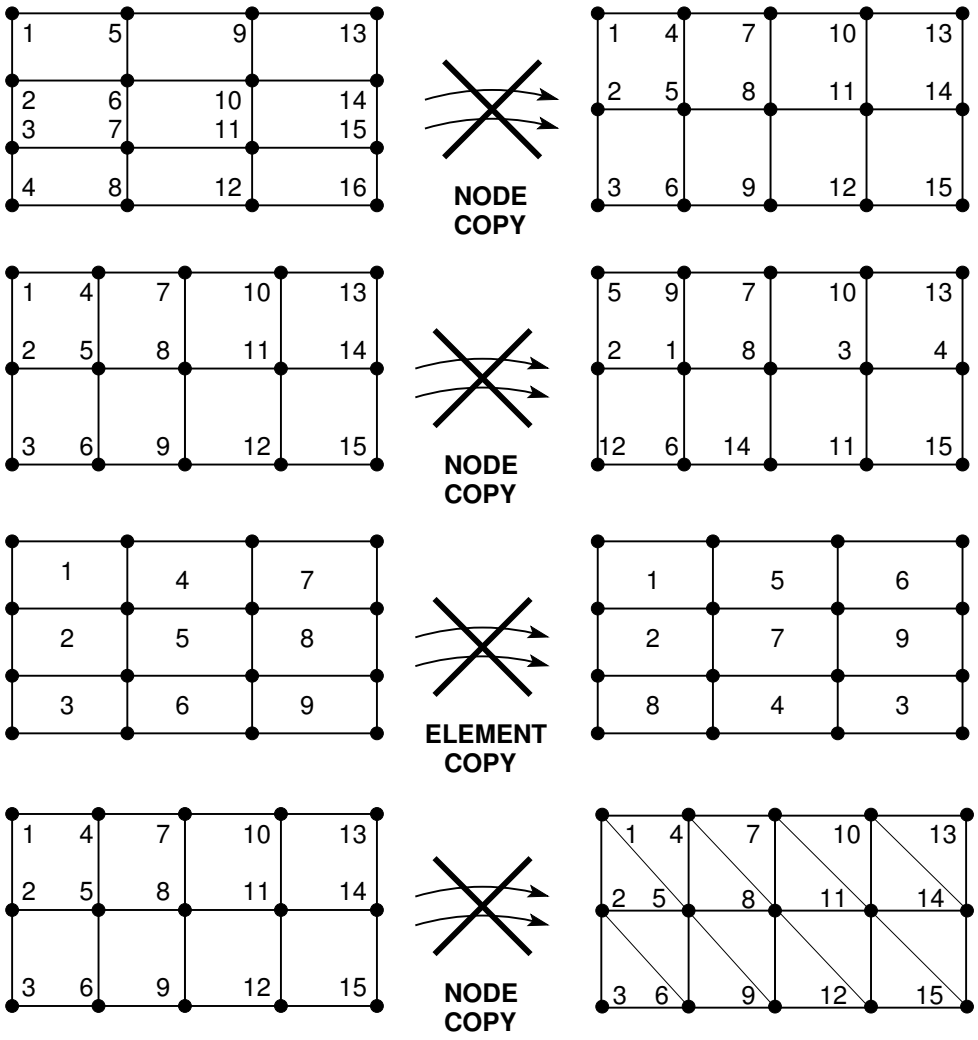


Figure 15.2. Invalid Transfer Operation

```
        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

    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 26.1 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

    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
```

15.2 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.

15.2.1 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.

15.2.2 All Fields

Scope: Transfer

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

15.2.3 Copy

Scope: Transfer

`Copy` *`Option1 Option2`* `From` *`From_region_name`* `To` *`To_region_name`*

Parameter	Value	Default
From_region_name	string	undefined
To_region_name	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.

15.2.4 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.

15.2.5 Exclude Ghosted

Scope: Transfer

Summary exclude ghosted nodes from a copy transfer

15.2.6 From

Scope: Transfer

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

15.2.7 Gauss Point Integration Order

Scope: Transfer

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

Parameter	Value	Default
Order	integer	undefined

Summary Integration order to use when transferring to Gauss points.

15.2.8 Interpolate

Scope: Transfer

`Interpolate` *`Option1 Option2`* `From` *`From_region_name`* `To` *`To_region_name`*

Parameter	Value	Default
From_region_name	string	undefined
To_region_name	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.

15.2.9 Interpolation Function

Scope: Transfer

`Interpolation Function` *`User_Subroutine`*

Parameter	Value	Default
User_Subroutine	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.

15.2.10 Nodes Outside Region

Scope: Transfer

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.

15.2.11 Search Coordinate Field

Scope: Transfer

`Search Coordinate Field` *`Source_field_name`* `State` *`Option1`* `To` *`Destination_field_name`* `State` *`Option2`*

Parameter	Value	Default
Source_field_name	`string`	`undefined`
Destination_field_name	`string`	`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.

15.2.12 Search Geometric Tolerance

Scope: Transfer

`Search Geometric Tolerance {=|are|is}` *`Geometric_tolerance`*

Parameter	Value	Default
Geometric_tolerance	`real`	`undefined`

15.2.13 Search Surface Gap Tolerance

Scope: Transfer

`Search Surface Gap Tolerance {=|are|is}` *`Surface_gap_tolerance`* `[ Or Less ]`

Parameter	Value	Default
Surface_gap_tolerance	`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 neccessary 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.

15.2.14 Search Type

Scope: Transfer

15.2.15 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.

15.2.16 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.

15.2.17 Send

Scope: Transfer

`Send` *`Predefined-transfer`* `Fields`

Parameter	Value	Default
Predefined-transfer	{}	undefined

Summary Use predefine transfer semantics provided by the specified name.

15.2.18 Send Block

Scope: Transfer

`Send Block` *`From_blocks...`* `To` *`To_blocks...`*

Parameter	Value	Default
From_ blocks	string...	undefined
To_ blocks	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

15.2.19 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
Source_field_ name	string	undefined
Destination_field_ name	string	undefined

Summary Specifies the mapping between source and destination field names. Vector and tensor fields can be subscripted using parenthesis and 1's based or brackets and 0 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 0 based with brackets or 1 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

Chapter 16

Time Integration Commands

16.1 Setting Up a Transient Problem

All Aria input files must include a Solution Control Description block in the Procedure section of the input file. The following example illustrates use of the solution control command block. This example demonstrates that time stepping for transient analysis can be carried out with either fixed 16.3 or adaptive time stepping 16.4.

```
.
.
.
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
   .
   .
   .
```

It is worth pointing out that for each named "Begin Transient" command block there should be a corresponding named "Begin Parameters for Transient" command block. This correlation establishes the mapping between a time block and its associated parameters. The intent of time block parameters is often misunderstood thus it is worth describing some of the underlying detail how the parameters are used.

16.2 Initial Time Step Estimation for Thermal Problems

The analysis of a transient diffusion problem requires the selection of a suitable time step for the integration procedure. This process can be automated with the time step being adaptively selected to preserve a specified time truncation error. Aria provides an adaptive time-stepping functionality, as described in Section 16.4. However, the time step selection process is not self-starting and some initial estimate of an appropriate time step is still required. In the following the discussion is primarily limited to implicit integration methods, since these methods are usually the methods of choice for most conduction models.

The selection of too *large* an initial time step can result in the loss of temporal accuracy in the solution and produce a nonphysical oscillatory response. Likewise, an inappropriately *small* initial time step may produce nonphysical, spatial oscillations in the early time temperature field due to the limited resolution ability (of temperature gradients) of the finite element mesh. Either of these difficulties may lead to stability problems if the boundary value problem is nonlinear. In any event, the solution during the oscillatory period is not accurate and these occurrences are to be avoided. A method for estimating an appropriate initial time step is outlined below. This procedure is originally due to Levi ([25]) though it has been discussed by several other authors ([26, 27, 28]).

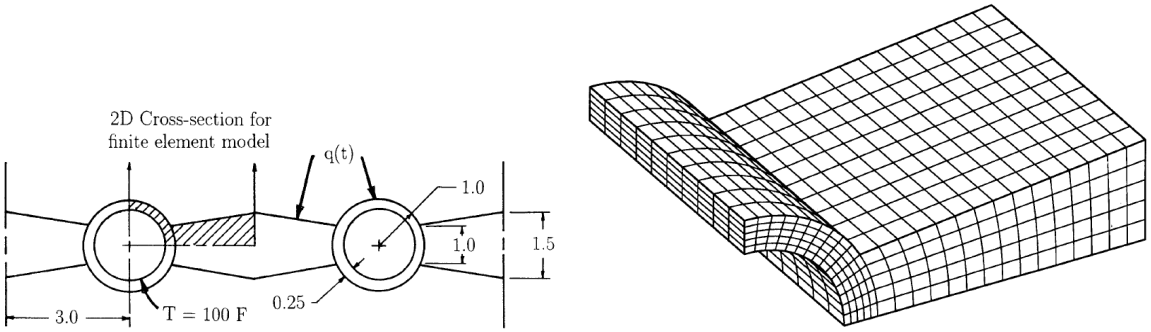
In the following development, it is assumed that a finite element mesh has been constructed that will adequately model the thermal phenomena of interest (e.g., thermal shock problems will require a fine mesh near a boundary while slower thermal transient will be less demanding on mesh refinement). For a given spatial discretization, a local characteristic length, Δx, is chosen based on element size. Typically, this characteristic length is measured normal to a boundary on which a temperature or heat flux (source) disturbance occurs. Based on the characteristic length, the local heat transfer coefficient, h, and the local thermal conductivity, k, a local element Biot number ($Bi = h\Delta x/k$) can be computed. Note that for a prescribed temperature, heat flux or heat source boundary condition, the heat transfer coefficient, h, is assumed to be large leading to a large Biot number.

In order to bound the thermal gradient that will occur at the boundary in the first time step, the ratio of the temperature at a distance Δx (characteristic length) from the boundary to the temperature on the boundary is selected. Let this ratio be defined by $\Theta = T(\Delta x)/T_{surface}$. Typical values of Θ will range from 0.10 to 0.25. With the estimated values for local Biot number and temperature ratio Θ, a local Fourier number ($Fo = \alpha\Delta t/\Delta x^2$) may be found from the charts in Figures 16.2 and 16.3. These graphs are based on an analytic solution for one-dimensional conduction with a convective boundary condition ([29]). A value of the local Fourier number and values for the characteristic length and thermal diffusivity, α, then allow a time step to by computed.

As an example of the procedure outlined above consider the transient problem described in the following example:

Consider the transient thermal response of a finned tube radiator, as shown in Figure 16.1(a). Thermal properties for the radiator are as follows: $\rho = 2.7071 \times 10^3$ kg/m^3, $C = 870.854$ J/kg·K, $k = 207.566$ W/m·K, and $\alpha = 8.806 \times 10^{-5}$ m^2/s. The characteristic size is based in the average size of the elements along the radiator fin, as shown in Figure 16.1(b), and is equal to $\Delta x = 3.175 \times 10^{-3}$ m. The boundary condition is an applied heat flux, so the Biot number is assumed large (infinite). Using a temperature ratio of $\Theta = 0.10$, then Figure 16.2 yields a Fourier number of $Fo \approx 0.20$. Thus,

$$\Delta t = \frac{(Fo)(\Delta x^2)}{(\alpha)} = \frac{(0.20)(3.175 \times 10^{-3})^2}{(8.806 \times 10^{-5})} = 0.023 \text{ seconds.} \tag{16.1}$$



(a) Schematic (b) Computational mesh

Figure 16.1. Geometry and mesh of finned radiator problem. Dimensions are in inches.

The above procedure can also be used to estimate the overall response time for a region. In this case the length scale is a characteristic length for the entire region and the temperature ratio should be order unity. The Fourier number will then produce the approximate time interval required to reach equilibrium.

16.3 Fixed Time Step Selection

Since Aria supports both adaptive and fixed time stepping it follows that this behavior be explicitly called out in the input file. For any given time block fixed time stepping is invoked via the combination of:
TIME STEP VARIATION = FIXED
INITIAL TIME STEP SIZE = *value*
PREDICTOR ORDER = 0
command lines within the Parameters for Transient command block. These command lines are further described in the adaptive time step selection section 16.4. Note that omission of the PREDICTOR ORDER = 0 command line for fixed time stepping will result in a default predictor error criteria to be considered in time step selection and may lead to unexpected results for presumed fixed time stepping scheme.

16.4 Adaptive Time Step Selection

Time step failure in Aria is defined as a Newton solution step that has not converged when the solution time was advanced. Aria time step selection is carried out by first addressing possible time step failures and then

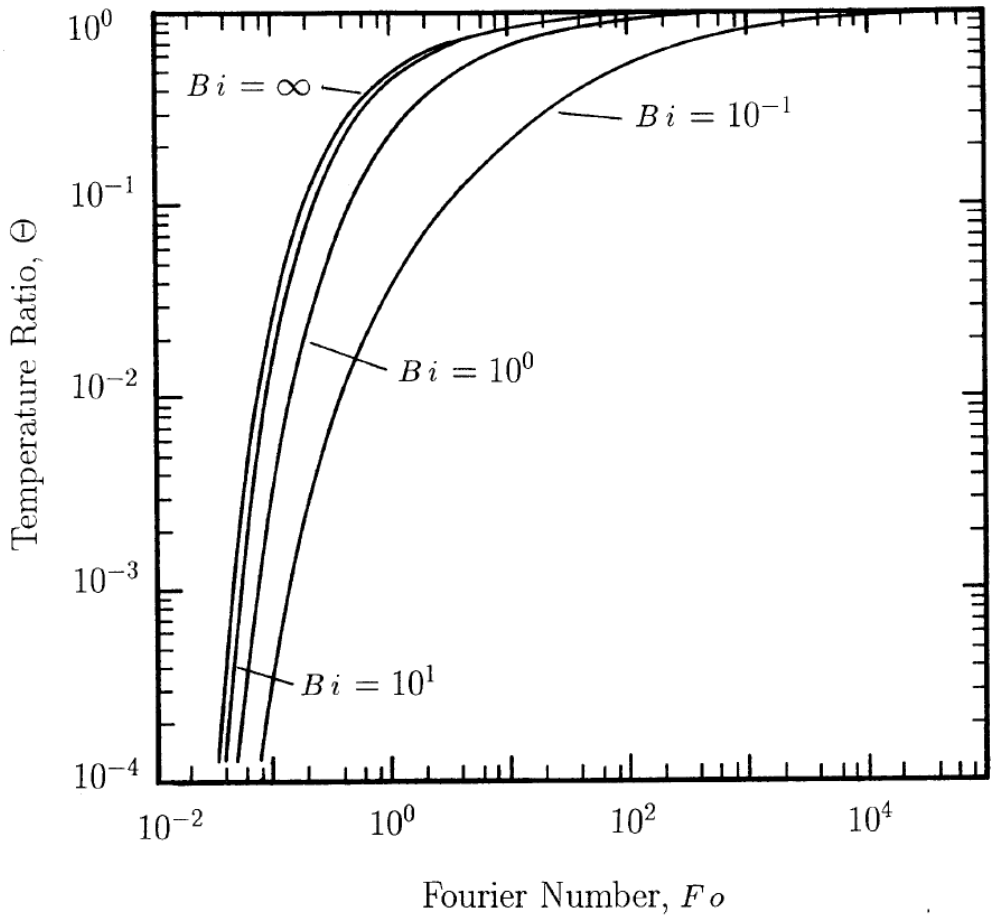


Figure 16.2. Temperature ratio versus Fourier number for various Biot numbers.

applying the type of time stepping requested, fixed or adaptive. If adaptive time stepping is requested then a number of criteria can considered when selecting the next time step with a primary objective of controlling solution error. Optimistically, use of the adaptive time step selection will result in a converged Newton step solution satisfying a user specified solution error tolerance and the solution time can again be advanced. Additional adaptive time stepping criteria are considered secondary relative to the error based criteria.

Foundationally, error based adaptive time stepping is based upon a solution error obtained using a predicted solution obtained from successful solves. Thus the time integration method often dictates how many solution steps into a simulation adaptive time stepping can begin. In most cases one must be able to obtain at least two successive solves using a fixed initial time step before adaptive time stepping begins. In many fluid simulations one routinely increases this number of constant steps to three or four using the PREDICTOR-CORRECTOR BEGIN AFTER STEP command line.

Certain model features like chemistry 36.2 will require that the phenomenological behavior within the current time step be considered when advancing the time step. We note that when adaptive time step selection is used, the subcycled chemistry timestep cannot be included directly in the adaptive time step selection as the chemistry time step will seldom satisfy the MINIMUM TIME STEP SIZE RATIO criteria 16.4.2. Here the minimum subcycled chemistry time step is modified to be comparable with other adaptive time step criteria 36.5. Additionally, the adaptive time step selection can also be overridden by user requests for output to occur at specific times.

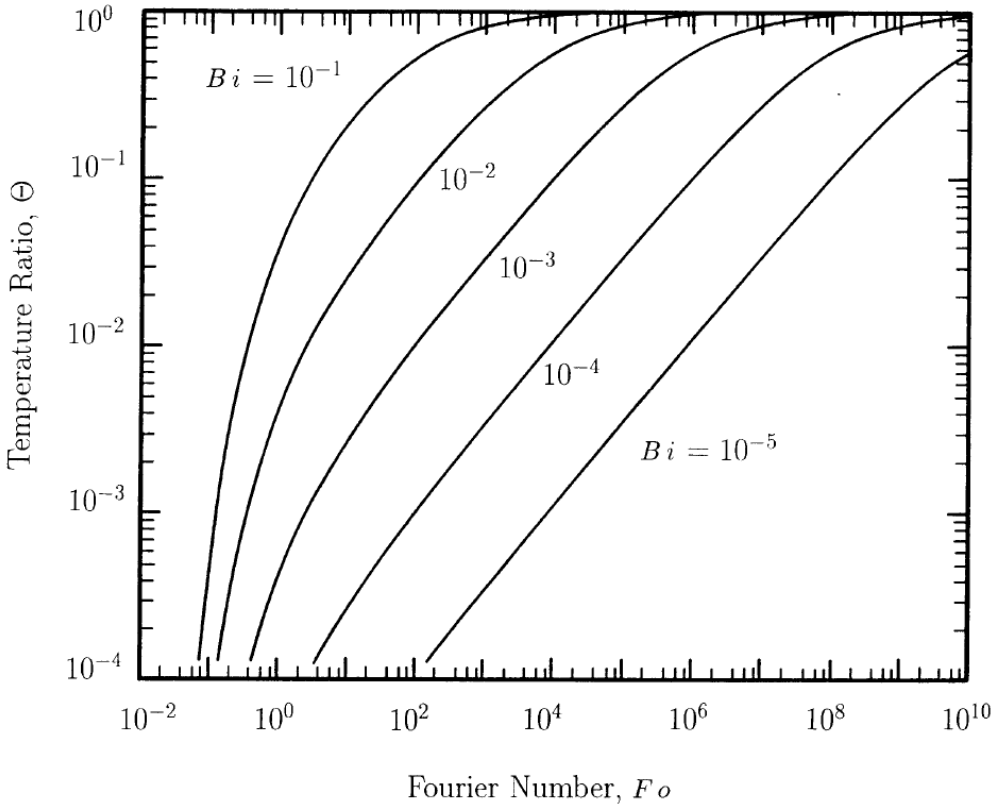


Figure 16.3. Temperature ratio versus Fourier number for various Biot numbers.

One of the quantities used in the selection of time steps is the time step size ratio $r = \Delta t_{new}/\Delta t_{old}$, the ratio of a proposed (new) time step to the current (old) time step. Experience tells us that for a first and second order integrator bounding the time step size ratio, $0.5 \leq r \leq 2.0$. While maintaining these bounds usually minimizes the time truncation error, a user is permitted to manually alter these bounds. For the BDF2 integrator the suggested theoretical bounds are $0.5 \leq r \leq 2.14$, where the upper bound is strictly enforced to ensure stability of the integrator. The approach used in Aria to control time stepping strictly relevant to physical phenomena such as phase change is to artificially modify r via a Phase Change Relaxation Factor.

16.4.1 Time Step Failure

Aria transient simulations are monitored for time step failure. For each solution time step we expect that the the specified nonlinear tolerance will be achieved in the specified number of nonlinear iterations within a Newton step. If the convergence condition is not met the time step is said to have "Failed" and solution time will not be allowed to advance unless the user has explicitly stated that failed time steps will be accepted with the (ACCEPT SOLUTION AFTER MAXIMUM NONLINEAR ITERATIONS = TRUE) command line. If the non-converged solution is not accepted, a failed time step counter is incremented, the time step is reduced by a factor determined using the *time step size ratio*. Similarly, when using adaptive time stepping the *time step size ratio* is also used to define upper bounds on the adaptive time step when the solution step is converged.

When using adaptive time stepping, a time step size ratio r is used to compute a new time step Δt_{n+1}

based upon the evaluation

$$r = \frac{\Delta t_{n+1}}{\Delta t_n}$$

where Δt_{n+1} is the minimum time step provided from the adaptive time step selection process 16.4.2 and Δt_n is the timestep in the current time step. When a time step fails the step will be repeated using a value defined using a FAILED TIME STEP SIZE RATIO (default 0.5) and the Newton step will be retried with the revised time step

$$\Delta t_{n+1} = r\ \Delta t_n \quad .$$

This process of updating Δt_{n+1} will be repeated until either a convergence condition is reached (convergence is achieved) or when the trial time step satisfies the FAIL TIME STEP WHEN TIME STEP SIZE RATIO IS BELOW criteria is reached (no convergence). In the former case the time step is advanced and in the latter case the simulation will be terminated.

During phase change, the model using the energy equation for temperature employs a mushy zone formulation hence it is important that the the temperature field values not bypass the range of the mushy zone so that the heat of fusion be accounted for. In this case an additional algorithm is used to detect when a nodal temperature has bypassed the mushy zone. When the mushy zone has been bypassed within a timestep, the Newton step will be marked as failed and the current time step will be scaled by a user specified PHASE CHANGE RELAXATION FACTOR in the same way that the failed time step size ratio is applied.

In some simulations a user may wish to interrupt the Newton step (fail the step) based upon the residual behavior. For many problems an aberrant behavior is recognized early on by noting the ratio of the current residual to the first residual. In cases where this ratio continues to grow it may be appropriate to terminate the Newton step and re-try with a different time step. This ratio can be specified by the user with the MAXIMUM LINEAR RESIDUAL RATIO.

16.4.2 Adaptive Time Step Selection

Adaptive time step selection consists of applying different criteria in the selection process. Each of the criteria are first described before linking them to the final selection procedure 16.4.3

Predictor Corrector

Let the predictor corrector tolerance, ϵ, be specified from the input file.

$$\frac{\Delta t_{n+1}}{\Delta t_n} = \left(b\ \frac{\epsilon}{d_{n+1}} \right)^m$$

For 1st order time integrator $m = 1/2$, $b = 2.0$. For 2nd order time integrator $m = 1/3$

$$b = 3 \left(1 + \frac{\Delta t_{n-1}}{\Delta t_n} \right)$$

$$d_{n+1} = \frac{1}{U_{max}} \left[\sum_{i=1}^{N} \left(U_{i(n+1)} - U^p_{i(n+1)} \right)^2 \right]^{1/2}$$

so that the timestep based on predictor criteria is

$$\Delta t^{Pred} = \Delta t_{n+1} = \Delta t_n \left(b\ \frac{\epsilon}{d_{n+1}} \right)^m \quad .$$

Max_CFL

In some instances the stability of a flow simulation can be governed by a Courant-Freidrichs-Levy (CFL) number, CFL_{max} for the flow. If a user specifies that a CFL number be be carried out over the model then a candidate time step can be computed as $\Delta t_{maxCFL} = CFL_{lim}\Delta t_n / CFL_{max}$, where CFL_{lim} is provided using the COURANT LIMIT command line. The CFL criteria possibly considered for Δt are

Δt_{maxCFL} standard fluid
Δt_{maxCFL} standard fluid phase A
Δt_{maxCFL} standard fluid phase B
Δt_{maxCFL} standard fluid phase C
Δt_{maxCFL} level set.

Minimum Timestep Based Upon MAXIMUM TIME STEP SIZE RATIO

A time step size ratio involving the future time step and the current time step is $r = \Delta t_{n+1}/\Delta t_n$. Letting r_{max} be the user specified maximum time step size ratio then the largest timestep achievable with r_{max} is $\Delta t = \Delta t_n * r_{max}$.

Minimum Timestep Based Upon MINIMUM ACCEPTABLE TIME STEP SIZE RATIO

A time step size ratio involving the adaptive time step and the current time step is $r = \Delta t_a/\Delta t_n$. Letting r_{min} be the user specified minimum acceptable time step size ratio then we require that $r > r_{min}$ else the time step is considered to be failed, the proposed time step is too small.

Maximum Solution Increment

Let the user specified maximum increment in any solution DOF be $(\Delta U)_{Max}$. From a previous successful solution we compute the maximum change in any DOF $(\Delta U)_{MaxSoln}$ and use it to compute an estimate of its corresponding maximum derivative in the previous time step as $\dot{U}_{max} = (\Delta U)_{MaxSoln}/\Delta t_n$. For a new time step $\hat{t}_n$ we might expect that the maximum value of U in the next step might reach $(\Delta \hat{U})_{max} = \hat{t}_n \dot{U}_{max}$. Now if $(\Delta \hat{U})_{max} < (\Delta U)_{Max}$ we retain $\hat{t}_n$ otherwise we propose a new time step $\Delta t_n = 0.95 * (\Delta U)_{Max}/\dot{U}_{max}$ with the hope that the solution increment $(\Delta U)_{Max}$ will not be exceeded in the next time plane. The same process is repeated for all solution DOF to produce a representative Δt_n for the given problem. Although adaptive time stepping with Solution Increment requires at least one successful solve here we require at least two successful solves since control of solution error is the primary objective.

16.4.3 Calculate Adaptive Timestep

We now proceed to select an adaptive time step t_a based upon the minimum value associated with:

- Δt^{Pred} 16.4.2
- Courant Limit, Δt_{cfl} 16.4.2
- Maximum Time Step Size, Δt_{max}
- Maximum Time Step Size Ratio, r_{max} 16.4.2
- Maximum Solution Increment 16.4.2
- Minimum Time Step Size
- Note: If chemistry is present, it will take precedence over all other candidates above

Let n_{tb} be the timeblock counter (step within the current timeblock).

Let n_{Pred0} be the user specified number of constant steps that begin any timeblock (default minimum is 2).

Let the current timestep be Δt_n (INITIAL TIME STEP SIZE when starting a simulation),
the adaptive timestep be $\Delta t_a = \text{Real\_Max}$, and
predictor corrector timestep be $\Delta t^{Pred} = \text{Real\_Max}$.
Let Δt_{max} be the user specified MAXIMUM TIME STEP SIZE and Δt_{min} the MINIMUM RESOLVED TIMESTEP SIZE.

To begin the timestep selection we first compute the desired predictor corrector timestep $\Delta \hat{t}^{Pred}$ (16.4.2) with the understanding that we wish to obtain a nominal value Δt^{Pred} which may not be the same as $\Delta \hat{t}^{Pred}$.

if $n_{tb} < n_{Pred0}$
  compute predictor error
  $\Delta t^{Pred} = \Delta t_n$
else
  if $\Delta \hat{t}^{Pred} < \Delta t_{min}$ then
    if $\Delta t_n > \Delta t_{min}$ then
      $\Delta t^{Pred} = \Delta t_{min}$
    else
      if $\Delta \hat{t}^{Pred} < \Delta t_n$ then
        $\Delta t^{Pred} = \Delta t_n$
      else
        $\Delta t^{Pred} = \Delta \hat{t}^{Pred}$.
  else
    $\Delta t^{Pred} = \Delta \hat{t}^{Pred}$.

Finally, the candidate t_a value must then satisfy the MINIMUM ACCEPTABLE TIME STEP SIZE RATIO, r_{min}, 16.4.2 to be accepted as a viable time step.

The overall time stepping procedure is demonstrated in the figure below 16.4.

Time integration is currently defined in the same way for the various physics contained within Aria. The command lines used to specify parameters used in that definition are described in what follows.

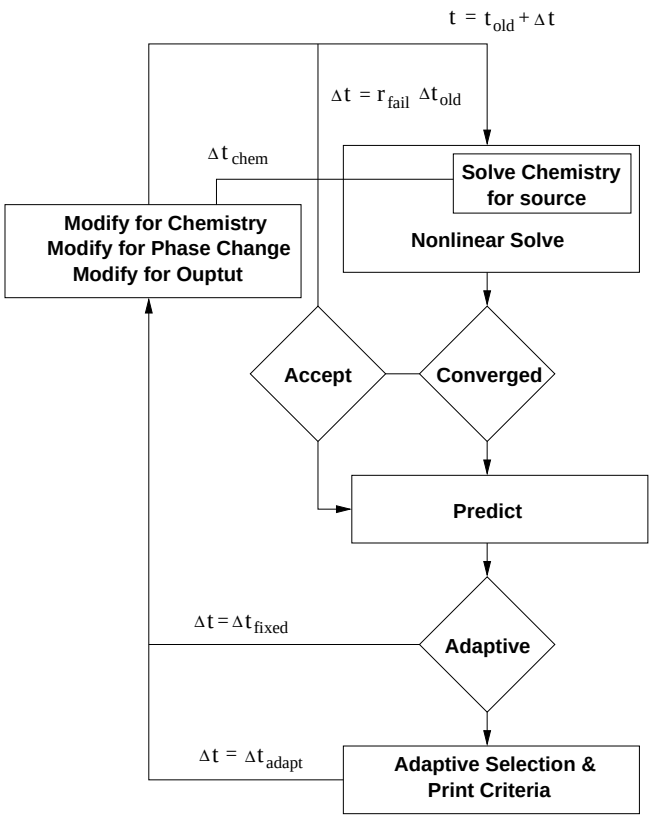


Figure 16.4. Time Step Selection Schematic.

16.4.4 Debugging Adaptive Time Step Failure

Debugging of time step failure involves straightforward interpretation of the log file output. The debugging process is best illustrated with an example of adaptive timestepping failure with the corresponding log file output which follows. Here the MINIMUM TIME STEP SIZE RATIO defaults to 0.5 and the previous timestep is Δt_n = 2.84E-01. After convergence of the solution the adaptive time step criteria are applied to obtain candidate time steps for the next time step. The minimum candidate time step is selected from those criteria hence Δt_{n+1} =3.661E-02 based upon the Predictor-Corrector Tolerance.

Using the Adaptive time stepping result the current time step size ratio is

$$r = \frac{\text{3.661E-02}}{\text{2.84E-01}} = 0.129 < 0.5 \ .$$

Since the current time step size ratio is less than the minimum time step ratio (0.5) the step cannot be advanced and the time step will be retried using a time step corresponding to the MINIMUM TIME STEP SIZE RATIO.

$$\Delta t_{n+1} = r\ \Delta t_n = (0.5)(\text{2.84E-01}) = \text{1.42E-01}$$

If one expects that time step should be reduced as solution time increases one may opt to abandon the default MINIMUM TIME STEP SIZE RATIO and set it to an appropriate lower value in the input file.

Note that failure of a time step due owing to violation of time step size ratio is an artifact of the adaptive time stepping selection criteria. Hence one should consider whether any of the selection criteria values can be modified by settings in the time step parameters command block.

```
-------------- For equation system name: main --------------
Global predictor error = 5.301e-05
Time step selection: dt <= 3.661e-02 (based on Predictor-Corrector Tolerance).
Time step selection: dt <= 2.839e+00 (based on Maximum Time Step Size Ratio).
Time step selection: dt <= 1.000e+03 (based on Maximum Time Step Size).
Time step selection: dt  = 2.512e+00 (Minimum Chemistry timestep block_284).
Time step selection: dt  = 3.661e-02 (Adaptive time stepping result).
Region::execute() time for my_region: 1.300e-02 sec.
Transient TimeBlock failed, step 284, time 3.0000e+01, time step 2.8394e-01
ERROR: Adaptive time stepping result yields a time step size ratio = 0.129
       which is less than the specified Minimum Time Step Size Ratio 0.5.
  Retrying step with a time step corresponding to time step size ratio of 0.5
```

In the approach shown, time step reduction is based upon relative change of the time step. Unfortunately, in problems where the adaptive time step drops dramatically at some point in the simulation the criteria previously described may cause the time step to fail repeatedly before the simulation step finally advances thus increasing the overall simulation time. In this case one can instead specify a lower bound on time step size ratio using the FAIL TIME STEP WHEN TIME STEP SIZE RATIO IS BELOW directive. Since the adaptive time stepping result is 0.129 then applying this approach to the example shown one could use

```
FAIL TIME STEP WHEN TIME STEP SIZE RATIO IS BELOW 0.1
```

to obtain more reasonable behavior.

16.5 Parameters For Aria Region

Scope: Parameters For

```
Begin Parameters For Aria Region RegionName

    Allow Order Switch For HowMany Steps
    Courant Limit {=|are|is} Courant_limit
    Fail Time Step On Bad Aztec Solver Status
    Fail Time Step When Time Step Size Ratio Is Below Ratio
    Failed Time Step Size Ratio {=|are|is} Ratio
    Initial Nonlinear Residual Tolerance For Time Step Control {=|are|is} tolerance
    Initial Time Step Size {=|are|is} Dt
    Interface Courant Limit {=|are|is} Interface Courant_limit
    Known Time Discontinuities {=|are|is} times...
    Limit Solution Increment Field_name {=|are|is} Value [ Action ]
    Maximum Acceptable Linear Residual Ratio {=|are|is} Ratio
    Maximum Consecutive Time Step Failures {=|are|is} NumFails
    Maximum Global Var Limit Encore_name {=|are|is} Value [ Action ]
    Maximum Linear Residual Ratio {=|are|is} Ratio
    Maximum Solution Limit Field_name {=|are|is} Value [ Action ]
    Maximum Time Step Size {=|are|is} Dt
    Maximum Time Step Size Ratio {=|are|is} Ratio
    Minimum Global Var Limit Encore_name {=|are|is} Value [ Action ]
    Minimum Resolved Time Step Size {=|are|is} Dt
```

```
Minimum Solution Limit Field_name {=|are|is} Value [ Action ]
Minimum Time Step Size {=|are|is} Dt
Predictor Order {=|are|is} order
Predictor-Corrector Begin After Step {=|are|is} Step
Predictor-Corrector Field Normalization {@|at|for|in|on|over} Field_name {=|are
|is} Norm_type [ Scaling ]
Predictor-Corrector Normalization {=|are|is} Normalization [ Scaling ]
Predictor-Corrector Tolerance {=|are|is} Predictor_corrector_tolerance
Reset Initial Time Step Size
Slope Of Time Step Size {=|are|is} slope
Stop When Initial Nonlinear Residual Is Below Tol
Time Integration Method {=|are|is} Time_integration_method
Time Step Variation {=|are|is} Time_step_variation
Use Initial Nonlinear Residual For Time Step Control {=|are|is} enable
End
```

Summary Defines region specific time stepping data

16.5.1 Allow Order Switch For

Scope: Parameters For Aria Region

```
Allow Order Switch For HowMany Steps
```

Parameter	Value	Default
HowMany	integer	undefined

Summary Allow the switching of time integration order from a higher order to first order.

Description This command allows for the switching of a higher order method to a first order method for a specified number of steps in order to improve the robustness of the simulation especially in the region of discontinuities such as chemistry shutoff. This command is mostly useful in the context of a SECOND_ORDER time integration method.

16.5.2 Courant Limit

Scope: Parameters For Aria Region

```
Courant Limit {=|are|is} Courant_limit
```

Parameter	Value	Default
Courant_limit	real	0.0

Summary The Courant Number limit. A value of 0.0 denotes INACTIVE.

16.5.3 Fail Time Step On Bad Aztec Solver Status

Scope: Parameters For Aria Region

Summary OPT to fail time the time step when the AZTEC solver status is -2 (Numerical Breakdown) or -3 (Loss of Precision).

16.5.4 Fail Time Step When Time Step Size Ratio Is Below

Scope: Parameters For Aria Region

`Fail Time Step When Time Step Size Ratio Is Below` *`Ratio`*

Parameter	Value	Default
Ratio	`real`	`undefined`

Summary Specify lower bound for adaptive time step size ratio failure criteria.

If the adaptive time step size ratio falls below this value, Aria will fail the time step. This criteria is specified in terms of the size of the time step size ratio, defined as dt_{n+1}/dt_n. When the adaptive time stepper senses the time step being cut back too much, the time step fails. The default is equal to the FAILED TIME STEP SIZE RATIO.

In simulations where the predicted time step is reduced substantially, a time step may fail repeatedly on this criterion. In that event the user may want to set this Ratio to a small positive value, possibly even zero.

16.5.5 Failed Time Step Size Ratio

Scope: Parameters For Aria Region

`Failed Time Step Size Ratio {=|are|is}` *`Ratio`*

Parameter	Value	Default
Ratio	`real`	`0.5`

Summary Specifies the factor used to cut the time step size when a step fails.

When a time step fails, the time step size is reduced by this factor.

16.5.6 Initial Nonlinear Residual Tolerance For Time Step Control

Scope: Parameters For Aria Region

`Initial Nonlinear Residual Tolerance For Time Step Control {=|are|is}` *`tolerance`*

Parameter	Value	Default
tolerance	`real`	`1.0`

Summary Set initial nonlinear residual tolerance for time step control.

Description Sets the tolerance on the initial nonlinear residual to be used for time step control.

16.5.7 Initial Time Step Size

Scope: Parameters For Aria Region

`Initial Time Step Size {=|are|is}` *`Dt`*

Parameter	Value	Default
Dt	`real`	`undefined`

Summary — Specifies the initial time step size for the given parameter block.

The initial time step size should always be specified for FIXED time stepping and for the starting time block of ADAPTIVE time stepping. When no other command lines affecting timestep are provided in the time step parameter block the time step will remain constant over the time extent of the parameter block.

16.5.8 Interface Courant Limit

Scope: Parameters For Aria Region

`Interface Courant Limit {=|are|is}` *`Interface Courant_limit`*

Parameter	Value	Default
Interface Courant_limit	`real`	`0.0`

Summary — The Interface Courant Number limit. Applies to both diffuse level set and CDFEM problems. A value of 0.0 denotes INACTIVE.

16.5.9 Known Time Discontinuities

Scope: Parameters For Aria Region

`Known Time Discontinuities {=|are|is}` *`times...`*

Parameter	Value	Default
times	`real...`	`undefined`

Summary — Specifies known time discontinuities in the solution that the adaptive time stepping procedure must capture.

Description — Allows user to specify certain times that the adaptive time stepper must solve at because the solution is known to have a discontinuity there. This is useful for problems where the boundary conditions change suddenly for example.

16.5.10 Limit Solution Increment

Scope: Parameters For Aria Region

`Limit Solution Increment` *`Field_name`* `{=|are|is}` *`Value`* `[` *`Action`* `]`

Parameter	Value	Default
Field_name	`string`	`undefined`
Value	`real`	`REAL_MAX`

Summary Provides user control for limiting maximum change in a chosen solution Field over a timestep. When action is PREDICT a time step estimation is computed based upon the predicted solution change. When action is FAIL_STEP the time step is still predicted but the the time step will also fail if the suggested solution increment is exceeded.

Description This command allows one to roughly specify the maximum magnitude that a solution Field can change within a step. A solution increment is estimated based upon the maximum change in the Field in the previous timestep and the proposed adaptive timestep. When the specified value of maximum solution increment is exceeded, the proposed adaptive timestep is reduced in an attempt to limit the solution increment.

One command line should be supplied for each solution Field one wishes to restrict. The restriction is applied directly to the solution Field values, no scaling is invoked.

16.5.11 Maximum Acceptable Linear Residual Ratio

Scope: Parameters For Aria Region

`Maximum Acceptable Linear Residual Ratio {=|are|is}` *`Ratio`*

Parameter	Value	Default
Ratio	`real`	`undefined`

Summary Specifies a failure criterion for a nonlinear step based on the ratio of the final linear residual to a nonlinear residual.

The ratio must be greater than zero and less than one.

16.5.12 Maximum Consecutive Time Step Failures

Scope: Parameters For Aria Region

`Maximum Consecutive Time Step Failures {=|are|is}` *`NumFails`*

Parameter	Value	Default
NumFails	`integer`	20

Summary Specifies a maximum consecutive of time step failures before a simulation is terminated. This prevents the application from continued cutback of the time step when recovery of the solution is unlikely.

16.5.13 Maximum Global Var Limit

Scope: Parameters For Aria Region

`Maximum Global Var Limit` *`Encore_name`* `{=|are|is}` *`Value`* `[` *`Action`* `]`

Parameter	Value	Default
Encore_name	`string`	`undefined`
Value	`real`	`undefined`

Summary This command allows the simulation to continue only when the given Encore variable is lower than a certain magnitude. Inclusion of the optional argument RETRY will cause the solution

step to fail and the time step to be reduced. This sequence will be carried out a maximum of three steps, after which the simulation will be terminated.

Description This command allows the simulation to continue only when the given Encore variable is lower than a certain magnitude.

16.5.14 Maximum Linear Residual Ratio

Scope: Parameters For Aria Region

`Maximum Linear Residual Ratio {=|are|is}` *`Ratio`*

Parameter	Value	Default
Ratio	`real`	`undefined`

Summary Specifies a failure criterion for a nonlinear step based on the ratio of a nonlinear residual to the initial nonlinear residual.

NOTE: The linear residual is not utilized with this check.

This criteria is not be used unless requested by the user.

16.5.15 Maximum Solution Limit

Scope: Parameters For Aria Region

`Maximum Solution Limit` *`Field_name`* `{=|are|is}` *`Value`* `[` *`Action`* `]`

Parameter	Value	Default
Field_name	`string`	`undefined`
Value	`real`	`REAL_MAX`

Summary Provides user control for limiting maximum value of a Field solution over a time step. Optional parameters are continue or terminate. When a step fails the step will be cut back and and the optional behavior is invoked when three consecutive fails have occurred. Default behavior is to terminate.

Description This command allows one to restrict the maximum magnitude that a solution Field can achieve within a simulation by causing the time step to fail when the Field value exceeds a user specified magnitude.

Oftentimes the maximum value will be tied to an event that occurs at a particular magnitude of the Field. This command can be used to impose a different solution behavior when this magnitude of the Field is realized by forcing a failure of the solution step. When the step is failed due to solution limit then cutback of the time step is allowed to occur for two consecutive steps. If the time step is cut back a third time in succession, the specified action of the command will be invoked.

By default the simulation will terminate when the maximum value is reached. If the optional argument is "continue" then time step failure will be over-ridden and the solution sequence will continue, presumably evolving toward lower magnitude values of the Field. This behavior may enable one to overcome large reductions in time step at discontinuous events which may occur for maximum values of the Field.

16.5.16 Maximum Time Step Size

Scope: Parameters For Aria Region

`Maximum Time Step Size {=|are|is}` *Dt*

Parameter	Value	Default
Dt	`real`	`REAL_MAX`

Summary

Specifies the maximum time step size for this parameter block.

The time step will not be allowed to exceed this value, regardless of adaptive time step selection and regardless of time step requests for other regions (for loosely coupled simulations).

16.5.17 Maximum Time Step Size Ratio

Scope: Parameters For Aria Region

`Maximum Time Step Size Ratio {=|are|is}` *Ratio*

Parameter	Value	Default
Ratio	`real`	`2.0/2.4`

Summary

Specifies the maximum allowable growth rate of the adaptive time step size.

If the time step selection computes a new time step size that exceeds the previous step size by more than this amount, then the new step size is limited to the product of this factor and the previous step size.

The default is 2.0 for FIRST_ORDER and SECOND_ORDER integration methods.

For BDF2 a value of 2.4 is imposed to enforce stability of the integrator.

16.5.18 Minimum Global Var Limit

Scope: Parameters For Aria Region

`Minimum Global Var Limit` *Encore_name* `{=|are|is}` *Value* `[` *Action* `]`

Parameter	Value	Default
Encore_name	`string`	`undefined`
Value	`real`	`undefined`

Summary

This command allows the simulation to continue only when the given Encore variable is higher than a certain magnitude. Inclusion of the optional argument, RETRY, will cause the solution step to fail and the time step to be reduced. This sequence will be carried out a maximum of three steps, after which the simulation will be terminated.

Description

This command allows the simulation to continue only when the given Encore variable is higher than a certain magnitude.

16.5.19 Minimum Resolved Time Step Size

Scope: Parameters For Aria Region

`Minimum Resolved Time Step Size {=|are|is} Dt`

Parameter	Value	Default
Dt	`real`	`0.0`

Summary

Specifies the minimum resolved time step size. If the time step size lies below this value, the step will not fail when the time step size ratio falls too low.

This setting serves as a floor value since the adaptive time stepper will not select a time step size below this value, and any predictor-corrector errors are neglected when the time step size falls below this value.

16.5.20 Minimum Solution Limit

Scope: Parameters For Aria Region

`Minimum Solution Limit Field_name {=|are|is} Value [ Action ]`

Parameter	Value	Default
Field_name	`string`	`undefined`
Value	`real`	`undefined`

Summary

Provides user control for limiting minimum value of a Field solution over a time step. Optional parameters are continue or terminate. When a step fails the step will be cut back and and the optional behavior is invoked when three consecutive fails have occurred. Default behavior is to terminate.

Description

By default the solution minimum limit for all Fields is not active. This command allows one to restrict the minimum magnitude that a solution Field can achieve within a simulation by causing the time step to fail when the Field value goes below a user specified value

See MAXIMUM SOLUTION LIMIT command for more info.

16.5.21 Minimum Time Step Size

Scope: Parameters For Aria Region

`Minimum Time Step Size {=|are|is} Dt`

Parameter	Value	Default
Dt	`real`	`0.0`

Summary

Specifies the minimum time step size for the given parameter block. The time step will not be allowed to fall below this value, regardless of adaptive time step selection and regardless of time step requests from other regions (for loosely coupled simulations).

16.5.22 Predictor Order

Scope: Parameters For Aria Region

`Predictor Order {=|are|is} order`

Parameter	Value	Default
order	integer	undefined

Summary

Specifies the order of the extrapolation used to predict the solution in transient simulations. An order of 0 corresponds to no extrapolation, instead copying the previous solution into the predicted solution.

Description

This command sets the order of extrapolation in the predictor for transient problems. A value of 0 basically turns off the extrapolation, and instead copies to old solution to the predicted solution.

The`PREDICTOR ORDER` defaults to 0 for problems if Gas Dynamics equations are defined. Otherwise, it is set to the order of the time integrator for the current time block i.e 1 for first order and 2 for second order

16.5.23 Predictor-Corrector Begin After Step

Scope: Parameters For Aria Region

`Predictor-Corrector Begin After Step {=|are|is}` *`Step`*

Parameter	Value	Default
Step	integer	2 / 3

Summary

Specifies after which solution step the Predictor-Corrector timestep calculation will begin to be used in computing an adaptive timestep. Predictor error will always be computed but will not actually be used until the prescribed step is reached.

Default value is 2 for first-order time integration and 3 for second-order time integration. Thus for first-order time integration, a candidate predictor-corrector timestep could appear in step 3.

16.5.24 Predictor-Corrector Field Normalization

Scope: Parameters For Aria Region

`Predictor-Corrector Field Normalization {@|at|for|in|on|over}` *`Field_name`* `{=|are|is}` *`Norm_type`* `[` *`Scaling`* `]`

Parameter	Value	Default
Field_name	string	undefined

Summary

Allows user to change the predictor-corrector normalization type for a given field variable separately from the global option.

Description

Allows user to change the predictor-corrector normalization type for a given field variable separately from the global option.

Normalization types are:

- NONE: Scale by unity
- MAX (default): Scale by the maximum field value
- USER: Scale by a constant (user-supplied) value

16.5.25 Predictor-Corrector Normalization

Scope: Parameters For Aria Region

Summary

Provides the ability to specify how the predictor-corrector error is normalized.

Currently supported values are MAX, NONE, and USER. If desired, L2 could be added easily. A possibly better method would be to allow user-specified scales for the variables but that's not yet supported.

The reason for allowing NONE (or even better, a user-specified scale) is to better handle problems that involve something that is zero-based with finite changes. This can happen in ALE problems where the adaptive time stepper is overly restrictive initially when the displacements are all nearly zero, but growing to some finite value.

Default is MAX.

16.5.26 Predictor-Corrector Tolerance

Scope: Parameters For Aria Region

`Predictor-Corrector Tolerance {=|are|is}` *`Predictor_corrector_tolerance`*

Parameter	Value	Default
Predictor_corrector_tolerance	`real`	`0.001`

Summary

Specifies the tolerance for the difference between the predicted solution and the implicitly solved corrector solution used sed in adaptive time step selection.

The adaptive time step selection formula is

$$\Delta t_{n+1} = \Delta t_n \left(b \frac{\epsilon}{d_{n+1}} \right)^m \tag{16.2}$$

where $\Delta t_n \equiv t_{n+1} - t_n$ is the time step size from the most recent solution, $\Delta t_{n+1} \equiv t_{n+2} - t_{n+1}$ is the new time step size, ϵ is the predictor-corrector tolerance and d_{n+1} is the norm of the difference between the predicted and actual solutions at time t_{n+1}. For first order time integration $m = 1/2$ and $b = 2$. For second order time integration $m = 1/3$ and $b = 3(1 + \Delta t_{n-1}/\Delta t_n)$. See [30].

16.5.27 Reset Initial Time Step Size

Scope: Parameters For Aria Region

Summary

Enforce use of initial time step size specification.

Description

When using adaptive time stepping with restart the initial time step size is taken from the restart file rather than the INITIAL TIME STEP SIZE specification. This choice of time step can be problematic if one wishes to change model features at the beginning of a restart. This command forces the time step selection to respect an initial time step size specification.

16.5.28 Slope Of Time Step Size

Scope: Parameters For Aria Region

`Slope Of Time Step Size {=|are|is}` *slope*

Parameter	Value	Default
slope	`real`	`undefined`

Summary Specifies the linear growth rate of the time step size for the given parameter block. The time step size will grow by this amount at each step unless limited by other factors.

16.5.29 Stop When Initial Nonlinear Residual Is Below

Scope: Parameters For Aria Region

`Stop When Initial Nonlinear Residual Is Below` *Tol*

Parameter	Value	Default
Tol	`real`	`undefined`

Summary Specify time block termination criteria based upon nonlinear residual value. If the initial nonlinear residual falls below this value, Aria will end the time block. This is useful for obtaining pseudo-transient solutions of steady-state problems.

16.5.30 Time Integration Method

Scope: Parameters For Aria Region

Summary Specifies the order of time integration.

Valid options are BACKWARD_EULER, FIRST_ORDER, SECOND_ORDER, BDF2, MIDPOINT_RULE and AVERAGE_ACCELERATION.

Note that the stability criteria of BDF2 will restrict the timestep growth by imposing a MAXIMUM TIME STEP SIZE RATIO of 2.14.

16.5.31 Time Step Variation

Scope: Parameters For Aria Region

Summary Specifies how the time step sizes are to be derived. Choose between FIXED and ADAPTIVE time step selection methods.

When the time step is ADAPTIVE the time step is determined using a predictor-corrector error criteria.

16.5.32 Use Initial Nonlinear Residual For Time Step Control

Scope: Parameters For Aria Region

Summary Enable use of initial nonlinear residual as an error estimate to control timestep for loosely coupled problems.

Description When enabled this option will treat the initial non-linear residual as a proxy for the error in region-region convergence for loosely coupled problems. The time step will be controlled using a predictor-corrector like algorithm.

16.6 Other Timestep Related Commands

Because Aria can solve for many solution Fields it is inappropriate to include Field specific commands within the time block parameters. Because interaction of time stepping with particular simulation physics is often required, these interactions are handled via command lines appearing at the Region scope as illustrated below.

```
.
.
.
Begin Procedure My_Aria_Procedure

   Begin Solution Control Description
     .
     time stepping related commands
     .
   End
   .
   .
   Begin Aria Region My_Region
    .
    .
    other physics timestep related commands
    .
    .
   End

End
.
.
```

16.6.1 Phase Change Relaxation Factor

Scope:

`Phase Change Relaxation Factor {=|are|is}` *Prf*

Parameter	Value	Default
Prf	`real`	`0.5`

Summary Allows a user to specify the phase change relaxation factor for ensuring that the current computation step does not miss a freeze or melt event. This factor must lie between 0.0 and 1.0. Over-relaxation or under-relaxation time step reduction factor when phase change occurs within a time step. Used in conjunction with a phase change specific heat material model.

16.7 Predictor Related Commands

Because Aria can solve for many solution Fields it is inappropriate to include Field specific commands within the time block parameters. Nevertheless, since interaction of time stepping with solution prediction is required, these interactions are handled via command lines appearing at the Region scope as illustrated below.

```
.
.
.
Begin Procedure My_Aria_Procedure

   Begin Solution Control Description
     .
     time stepping related commands
     .
   End
   .
   .
   Begin Aria Region My_Region
    .
    .
    predictor related commands
    .
    .
   End

End
.
.
```

16.7.1 Display Truncation Error For

Scope:

`Display Truncation Error For` *`Solution_dof`* `[ With` *`HowMany`* `Entries ]`

Parameter	Value	Default
Solution_dof	string	undefined

Summary — Provides ability to request log file output of the maximum computed truncation error values.

Description — This command allows one to request the maximum magnitudes of predictor error at a small number of DOF to the log file. For vector Fields only the magnitude over all components is considered. Note that additional log file output can be IO intensive so use of this command line should be restricted to debugging of a model.

Default is 5 values.

16.7.2 Maximum Number Of Rebalances

Scope:

`Maximum Number Of Rebalances {=|are|is}` *`Count`*

Parameter	Value	Default
Count	`integer`	`undefined`

Summary Specify the maximum number of times to perform a mesh rebalance.

16.7.3 Predictor Fields

Scope:

`Predictor Fields {=|are|is}` *`Field_names...`*

Parameter	Value	Default
Field_names	`string...`	`undefined`

Summary Specification of active fields to be used in predictor algorithm.

Description Specifies which fields to examine or ignore in the algorithm for adaptive time step selection. Fields that are *not* predictor fields will not be predicted solutions for the first two time steps. This is important for fields like pressure and electrostatic potential which may exhibit large jumps in their solutions between the initial conditions and the first solutions owing to the absence of time dependent terms in their governing equations during startup.

The selected fields will contribute to the d_{n+1} norm discussed in the `TIME TRUNCATION ERROR` time block command. This command line can be provided multiple times with cumulative results.

By default all fields are included. All field names following the optional "NOT" keyword will be excluded from the selection algorithm.

Chapter 17

Nonlinear Solution Strategy

17.1 Equation System Overview

In coupled physics problems one is tasked with solving a set of nonlinear coupled equations and ideally one would solve these equations as a fully-coupled monolithic system. In practice solving the set of equations monolothically may prove impractical owing to large linear system memory requirements and difficulty in locating a small radius of convergence. Thus in some cases one may wish to perform a segregated solve of the equations where each set of equations can be solved in a slightly different manner. This ability to solve subsets of the entire equation set is supported through a feature called Equation System by which criteria for solving a set of nonlinear equations can be applied to a select group of equations.

By default a single set of coupled equations is treated internally as an Equation System without need of additional input syntax. Here the solution strategy for the Equation System is solely determined by a collection of Nonlinear Solution Specifications 17.3. A typical setup for a single equation system might be

```
Begin Aria Region myRegion
  .
  Use Linear Solver solve_temperature
  .
  eq energy for temperature On block_1 using q1 with lumped_mass diff
  .
  nonlinear solution strategy = newton
  use dof averaged nonlinear residual
  maximum nonlinear iterations = 20
  nonlinear residual tolerance = 1.e-6
  .
  .
End Aria Region myRegion
```

Simulations in which different subsets of the coupled equations are solved in a segregated manner require that each subset be explicitly defined as an Equation System with a corresponding solution strategy which may differ for each equation system. When multiple Equation Systems are present, the overall segregated solution is governed by iteration and convergence criteria at a global/outer level 17.2.

A typical setup for segregated solution of multiple equation systems might be

```
Begin Aria Region myRegion
  .
  Segregated global nonlinear residual tolerance = 6.0e-9
  .
  Begin Equation System temperature
    .
    eq energy for temperature On block_1 using q1 with lumped_mass diff
    Use Linear Solver solve_temperature
    .
    nonlinear solution strategy = newton
    use dof averaged nonlinear residual
    maximum nonlinear iterations = 20
    nonlinear residual tolerance = 1.e-6
    .
  End Equation System temperature
  .
  Begin Equation System species
    .
    eq species for species_0 On block_1 using q1 with diff
    .
    Use Linear Solver solve_species
    .
    nonlinear solution strategy = newton
    use dof averaged nonlinear residual
    maximum nonlinear iterations = 10
    nonlinear residual tolerance = 1.e-8
    .
  End Equation System temperature
  .
End Aria Region myRegion
```

17.2 Global Solution For Equation Systems

The global solution for a set of Equation Systems is controlled by a set of segregated solve criteria. A simulation will advance to the next time step when these criteria are satisfied.

17.2.1 Segregated Global Minimum Convergence Rate

Scope:

`Segregated Global Minimum Convergence Rate {=|are|is}` *`MinRate`* `[ Number Of Steps {=|are|is}` *`MinSteps`* `]`

Parameter	Value	Default
MinRate	`real`	`undefined`

Summary Set a minimum convergence rate based on the initial nonlinear residual of each equation system in the problem. At each segregated iteration the equation system with the largest initial residual is tested using $r_N/r_{N-MinSteps} < MinRate^{M}inSteps$. If that condition is not met then the time step fails and is retried with a shorter dt.

17.2.2 Segregated Global Nonlinear Initial Residual Tolerance

Scope:

`Segregated Global Nonlinear Initial Residual Tolerance {=|are|is}` *`Pt`*

Parameter	Value	Default
Pt	`real`	`undefined`

Summary Convergence tolerance for global segregated nonlinear residual for the iteration. The iteration is considered finished when all of the individual equation systems have an initial nonlinear residual less than the given tolerance. This is cheaper than other tests that require recomputing the nonlinear residual. However, care should be taken since the initial nonlinear residual of the individual equation systems might not be a good measure of the final nonlinear residual of those systems due to deltas produced by other equation systems.

17.2.3 Segregated Global Nonlinear Number Of Steps

Scope:

`Segregated Global Nonlinear Number Of Steps {=|are|is}` *`Number_of_steps`* `[ Fail Step {=|are|is}` *`failValue`* `]`

Parameter	Value	Default
Number_of_steps	`integer`	`undefined`

Summary Maximum number of segregated solution steps within a time step. Depending on the failValue the solution will either be accepted, or the time step will fail when the maximum number of steps is reached.

17.2.4 Segregated Global Nonlinear Relative Residual Tolerance

Scope:

`Segregated Global Nonlinear Relative Residual Tolerance {=|are|is}` *`Pt`*

Parameter	Value	Default
Pt	`real`	`undefined`

Summary Convergence tolerance for global segregated nonlinear residual for the iteration. Satisfaction of this criterion is sufficient for the iteration to be considered finished. This line command can be used when the user desires to enforce a global relative residual tolerance for the entire segregated system that may be different from the individual equation systems value

17.2.5 Segregated Global Nonlinear Residual Tolerance

Scope:

`Segregated Global Nonlinear Residual Tolerance {=|are|is}` *Pt* `[ Minimum Steps {=|are|is}` *minStepsValue* `]`

Parameter	Value	Default
Pt	`real`	`undefined`

Summary Convergence tolerance for global segregated nonlinear residual for the iteration. Satisfaction of this criterion is sufficient for the iteration to be considered finished. This line command can be used when the user desires to enforce a global residual enforcement for the entire segregated system that may be different from the individual equation systems value

17.2.6 Segregated Global Nonlinear Target Number Of Steps Per Time Step

Scope:

`Segregated Global Nonlinear Target Number Of Steps Per Time Step {=|are|is}` *target_iterations*

Parameter	Value	Default
target_iterations	`integer`	`undefined`

Summary Set a target number of segregated iterations per time step to use for time step control. If more than the specified number of iterations are required to converge a time step the next time step size will be decreased proportional to the ratio of the target number and the number taken. If a maximum number of allowed segregated iterations is set it is recommended to set the target to 1-2 fewer iterations to avoid an excessive number of failed time steps.

17.3 Nonlinear Solution Specifications

The general solution strategy used in Aria is to solve the requested system of equations for solution variables ϕ incrementally. In the context of a Newton formulation then within the Newton step one will solve

$$J\Delta\phi = -R(\phi)$$

where J is the Jacobian matrix, $\Delta\phi$ is the solution increment and $R(\phi)$ is the system residual for the governing equations described in Chapter 5. This chapter describes basic nonlinear solution command line settings that influence how a converged solution will be obtained. For transient solutions these settings are used in combination with time integration settings described in Chapter 16 for advancing a solution to the next time step.

It is instructive here to review a typical log file output of the Aria solution summary in order to establish how a user might influence the solution procedure.

```
Equation System AriaRegion->main:
 * Step   : Transient, Strategy: NEWTON, Time: 2.20e+03, Step: 5.29e+00
 * Matrix: Solver: "Solve_Temperature", Unknowns: 1360, Nonzeros: 11578
 * Mesh   : Processor 0 of 1: 1253 of 1253 elems, 1363 of 1363 nodes
 * Computing View Factors for enclosure space
  N O N L I N E A R                L I N E A R
----------------------   -------------------------------------
Step  Resid    Delta     Itns Status  Resid    Asm/Slv Time
---- -------- --------   ---- ------ -------- ---------------
1    6.37e-02 1.10e+01   59    ok    8.23e-07 7.0e-03/2.0e-03
2    1.10e-04 8.18e-03   58    ok    1.19e-09 7.0e-03/2.0e-03
3    3.06e-07 8.81e-05   61    ok    3.68e-12 6.0e-03/2.0e-03
Termination reason: 8.80515e-05 < nonlinear_correction_tolerance(0.001)
```

In the log file output shown above, columns under the `N O N L I N E A R`, heading are related to the command lines described in this chapter. The columns `Resid` and `Delta` represent L2 norms of an aggregate residual $R^*(\phi)$ and solution increment vector $\Delta\phi$ while the column denoted `Step` denotes the iteration within this Newton step. Columns under the `L I N E A R` heading represent the number of linear solver iterations (`Itns`), the `Resid` column represents the linear system residual $R^*(\Delta\phi)$. Here $R^*(\Delta\phi) = J\Delta\phi + R(\phi)$, the residual for the assembled system of equations, is solved for $\Delta\phi$ to the tolerance requested in the Linear Solver specifications 20. The column labeled `Status` indicates whether the linear solve completed successfully, `ok` or `fail`, within the allowable number of iterations defined in the Linear Solver specifications. The column labeled `Asm/Slv Time` contain assembly and solution times for the system of equations.

Given a current value of ϕ then within each Newton step the residual $R^*(\Delta\phi)$ is first evaluated using $\Delta\phi = 0$ and tested for convergence. Here the convergence conditions are satisfied when either the `NONLINEAR RESIDUAL TOLERANCE` or `NONLINEAR CORRECTION TOLERANCE` are achieved. If the solution is not yet converged, the system of equations is turned over to the equation solver where it is solved for a new $\Delta\phi$ subject to the Linear Solver specifications. Then the Solver solution is used to update the nonlinear solution $\phi = \phi_{old} + \Delta\phi$ and the process begins anew subject to a user supplied `MAXIMUM NONLINEAR ITERATIONS`.

Within the Newton step a solution is said to `FAIL` if it is not converged within `MAXIMUM NONLINEAR ITERATIONS` and this occurrence will be noted in the log file. For transient simulations one will normally repeat the solution procedure with a smaller time step as described in Chapter 16. In some cases the user may chose to override the `FAIL` condition using the `ACCEPT SOLUTION AFTER MAXIMUM NONLINEAR ITERATIONS` command line but *one is cautioned that this option should be used with discretion as it may discredit any expectation of accuracy in transient results.* For completeness, a summary of pass, fail, and fail but accepted steps are included at the end of the log file.

For most applications the default `NONLINEAR RESIDUAL TOLERANCE` (1e-6) is adequate when accompanied by a Linear Solver specification of `RESIDUAL NORM TOLERANCE` minimum of 1.0e-8. Inclusion of the command line `USE DOF AVERAGED NONLINEAR RESIDUAL` is highly recommended, particularly when performing uniform mesh refinement studies. A summary of these and other nonlinear solution related command lines is given in what follows.

17.3.1 Accept Solution After Maximum Nonlinear Iterations

Scope:

`Accept Solution After Maximum Nonlinear Iterations {=|are|is}` *`Bool`*

Parameter	Value	Default
Bool	`{false|true}`	`FALSE`

Summary — Specifies whether solution is considered to be converged after MAXIMUM NONLINEAR ITERATIONS even if none of the convergence criteria are satisfied.

17.3.2 Filter Nonlinear Solution

Scope:

`Filter Nonlinear Solution For` *`AssociatedDoF FilterType`* `{=|are|is}` *`Values...`*

Parameter	Value	Default
AssociatedDoF	`string`	`undefined`
FilterType	`string`	`undefined`
Values	`string...`	`undefined`

Summary — Restrict the value of a solution variable used in the nonlinear solver iterations of a solution step.

Description — The nonlinear solution can be restricted to a range with upper and lower limits using the FILTER_TYPE = RANGE while supplying two bounding values. To set an upper or lower bound use FILTER_TYPE = MAXIMUM or FILTER_TYPE = MINIMUM while supplying a single bounding value.

17.3.3 Maximum Accepted Nonlinear Correction

Scope:

`Maximum Accepted Nonlinear Correction {=|are|is}` *`mnlc`*

Parameter	Value	Default
mnlc	`real`	`1e16`

Summary — Maximum allowed nonlinear correction tolerance during residual acceptance.

17.3.4 Maximum Accepted Nonlinear Residual

Scope:

`Maximum Accepted Nonlinear Residual {=|are|is}` *mnlr*

Parameter	Value	Default
mnlr	`real`	1e16

Summary Maximum allowed nonlinear residual tolerance during nonlinear correction acceptance.

17.3.5 Maximum Line Search Steps

Scope:

`Maximum Line Search Steps {=|are|is}` *MaxSteps*

Parameter	Value	Default
MaxSteps	`integer`	10

Summary Maximum number of line search steps per nonlinear iteration.

Description Only applicable when the LINE_SEARCH nonlinear solution strategy is used.

17.3.6 Maximum Nonlinear Iterations

Scope:

`Maximum Nonlinear Iterations {=|are|is}` *Np*

Parameter	Value	Default
Np	`integer`	20

Summary Number of allowable nonlinear iterations in one linear solution step.

Description The solution step will terminate when the number of nonlinear iterations are exceeded. Default value = 20.

17.3.7 Minimum Nonlinear Solves

Scope:

`Minimum Nonlinear Solves {=|are|is}` *Np*

Parameter	Value	Default
Np	`integer`	1

Summary Minimum number of actual nonlinear solves that are required.

Description Each nonlinear iteration involves first assembling the nonlinear residual and Jacobian matrix and second solving the linear system for a nonlinear correction that is applied to the estimate of the nonlinear solution.

The nonlinear solver will continue iterating until this minimum number of solves are performed. The default value is one (1) but setting this to zero can be useful in some situations.

17.3.8 Nonlinear Correction Ratio Tolerance

Scope:

`Nonlinear Correction Ratio Tolerance {=|are|is}` *Pt*

Parameter	Value	Default
Pt	`real`	`undefined`

Summary Convergence tolerance for the nonlinear correction. It is satisfied if the nonlinear correction is less than this specified multiple of the initial nonlinear correction.

17.3.9 Nonlinear Correction Scaling

Scope:

Summary Specifies how to scale the nonlinear correction norm.

Description NONE (default) - The unscaled nonlinear corrections will be used.

DOF_AMAX (recommended) - The corrections for each equation will be scaled by max(1., abs(max(solution_N)))

17.3.10 Nonlinear Correction Tolerance

Scope:

`Nonlinear Correction Tolerance {=|are|is}` *Pt*

Parameter	Value	Default
Pt	`real`	`1.0e-6`

Summary Convergence tolerance of nonlinear correction norm for the iteration. Satisfaction of this criterion is sufficient for the iteration to be considered finished.

17.3.11 Nonlinear Relaxation Factor

Scope:

`Nonlinear Relaxation Factor {=|are|is}` *Prf*

Parameter	Value	Default
Prf	`real`	`1.0`

Summary Weighting factor for fraction of a new nonlinear solution that will be applied to the linear solution update. Weighting factor must lie in the range 0.0 to 1.0.

17.3.12 Nonlinear Residual Minimum Convergence Rate

Scope:

`Nonlinear Residual Minimum Convergence Rate {=|are|is}` *`MinRate`* `[ Number Of Steps {=|are|is}` *`MinSteps`* `]`

Parameter	Value	Default
MinRate	`real`	`undefined`

Summary Set a minimum convergence rate based on the nonlinear residual At each nonlinear iteration the condition $resid_N/resid_{N-MinSteps} < MinRate^{MinSteps}$. If that condition is not met then the time step fails and is retried with a shorter dt.

17.3.13 Nonlinear Residual Ratio Tolerance

Scope:

`Nonlinear Residual Ratio Tolerance {=|are|is}` *`Pt`*

Parameter	Value	Default
Pt	`real`	`0.0`

Summary Convergence tolerance for the ratio of the nonlinear residual to the initial nonlinear residual for the solution iteration. Convergence is satisfied if the ratio is less than this specified multiple of the initial nonlinear residual.

17.3.14 Nonlinear Residual Scaling

Scope:

Summary Specifies how to scale the nonlinear residuals.

Description NONE - The unscaled nonlinear residuals will be used.

DOF_AMAX (recommended) - The residuals for each equation will be scaled by max(1., abs(max(solution_N)))

This option will only affect problems using Tpetra-based linear solvers.

17.3.15 Nonlinear Residual Tolerance

Scope:

`Nonlinear Residual Tolerance {=|are|is}` *`Pt`*

Parameter	Value	Default
Pt	`real`	`1.0e-6`

Summary Convergence tolerance of nonlinear residual for the iteration. Satisfaction of this criterion is sufficient for the iteration to be considered finished.

17.3.16 Nonlinear Solution Strategy

Scope:

Summary Specifies the nonlinear solution strategy.

Description Nonlinear solution strategy must be NEWTON, NEWTON_FINITE_DIFFERENCE or POINT_IMPLICIT. There is no default value so this line command is required.

For POINT_IMPLICIT one must also set: ACCEPT SOLUTION AFTER MAXIMUM NONLINEAR ITERATIONS = TRUE and MAXIMUM NONLINEAR ITERATIONS = 1

17.3.17 Use Dof Averaged Nonlinear Residual

Scope:

Summary Select whether to use a nonlinear residual normalized by the number of degrees of freedom instead of the raw global residual reported by the linear system.

Description Specifies that the reported nonlinear residual is scaled (divided by) the number of DOF in the problem. In selecting appropriate values for the NONLINEAR RESIDUAL TOLERANCE users are cautioned that the average nonlinear residual reported values may be several orders of magnitude smaller than when using the global nonlinear residual. This option is provided to allow comparison with residual values from other applications.

Chapter 18

Writing User Plugins

18.1 About Plugins

Users are free to extend Aria's library of material models, constitutive equations, boundary conditions, distinguishing conditions and source terms through the use of *plugins*. In order to do this, a user writes the C++ code to implement an Expression class. Thus *plugins* should not be confused with *user_subroutines* 33.20. Before delving into Aria plugins it's worth reading section 43.2 and scanning section 43.5 for an introduction to Aria's Expression system.

When a user supplies their own plugin it becomes a legitimate piece of Aria. Plugins have no performance penalty over Aria's built-in functionality and plugins have no added restrictions over built-in Expression regarding what can and can't be done. In fact, taking a user plugin and adding it to Aria proper (so that it becomes "owned" by Aria) is a fairly straightforward process.

It's worth re-stating here that since Aria employs a Newton formulation, many of the algorithms make use of DOF sensitivities, i.e., Aria often needs to know the derivative of your model with respect to all of the unknowns in the problem. There are three ways to supply the sensitivities in Aria: manually code them, use a numerical finite-difference function, or use (forward) automatic differentiation (FAD). The FAD method is ideal for plugins because the sensitivities are are performed analytically and exact, though there may be a small performance degradation. However, if only a few models use FAD there may be no measurable performance penalty. For that reason, we've designed our plugin system to use FAD by default. If one has different needs, please contact the user support group sierra-helpsandia.gov for help.

18.2 Compiling and Using Plugins with Dynamically Loaded Libraries

The easiest and most efficient way to compile and use plugins with Aria is to compile them into a dynamically loaded library that a pre-built aria executable is able to load at runtime.

18.2.1 Compiling a Plugin into a Shared Library

The easiest way to compile your plugin into a share library (`.so` file) is to use the `jamsub` tool. Within SNL, this tool is normally located in the `/sierra/Dev/tools/` directory. It's pretty simple to use:

```
$ jamsub -P aria -sierradir /path/to/sierra/project
```

If you happen to be working in a Sierra project that contains the Aria you'd like to build against then one can skip the `-sierradir` option. You can use the `-s` option to override the system for which you want to build; see the help (`-h`) for more options.

This command will find all of the `.C` files in the current directory and compile them into a file of the same name but with the `.so` extension. Note that you do not have to put your `.C` files in the Aria source directories – they can live with your problem input files in any working directory.

In the event one is not building on a standard Linux machine, the command line must also reference the target platform as follows:

For code versions before 4.26,

```
$  module load sierra
$  jamsub -P aria -sierradir /path/to/sierra/project/target_code_version
```

this will compile all plugins in the current directory.

For code versions 4.26 and beyond,

```
$  module load sierra-devel
$  module load sierra-apps/target_code_version
$  sierra -make aria -i input_filename.i
```

here the input file is first scanned and only the needed plugins will be compiled.

Running the above command line with an invalid platform name will provide the available platforms for which one can build an Aria executable.

18.2.2 Pointing to a Plugin form your Aria Input File

```
Begin Sierra My_Sierra_Job
   Load User Plugin file ./My_Density_Plugin.so
   Load User Plugin file ./My_Viscosity_Plugin.so
   .
   .
   .
```

18.2.3 Running Aria with Dynamically Loaded Plugins

The Aria executable is built to accommodate dynamically loaded libraries. Execution when using plugins is handled in the same way as when plugins are not being used:

```
$  sierra aria -i use_my_density.i -x /path/to/sierra/project
```

18.3 An Important Note About Model Names

In order to avoid name clashes, model names always end with the generic name of the quantity they provide. So, density model names always end in `_DENSITY`, viscosity model names end in `_VISCOSITY`, source model names end in `_SOURCE`, etc. For material models, it's easy to know what that ending is because it's the same as the left-hand side of the "=" sign in the material input block (with spaces replaced by an underscore "_").

Clearly, it's important to keep this in mind when naming and referring to your plugin model or else Aria won't be able to find it properly.

In this example, we'll name our density `MY_MODEL_DENSITY` (recall, it must end in `_DENSITY`) so in the input file we'll have to refer to it as `MY_MODEL` since the ending is automatically added.

18.4 The Input File

As of version 4.48 there is a difference between referencing plugins for material models vs source terms or boundary conditions. Source or BC plugins can be referenced in the input file similar to the way Aria's built-in Expressions are referenced. The name of the model is just whatever your plugin is named without the file extension, e.g., `MY_MODEL`. Material model plugins use the User_Plugin model type and provide the plugin name as the name parameter.

```
Begin Aria Material Kryptonite
   Density = User_Plugin name=My_Model a=1.0 b=-0.01
   ...
End

...

Begin Aria Region Region
  ...
  Source for Energy on block_1 = My_Source_Plugin

  BC flux for Energy on surface_1 = My_Flux_Plugin d=5.
  ...
End
```

It's important to note that the plugin name must not conflict with model names used internally by Aria (the outcome would be, at least to users, ambiguous). Aria will verify this and produce an error if there's a name clash.

18.5 Example Plugin Code: My_Density.C

In this section we'll write the code to supply a density function which is a cubic polynomial in temperature (T),

$$\rho = a + bT + cT^2 + dT^3.$$

Normally, writing C++ code requires using a header (`.h`) file and an implementation (`.C`) file but since no other code needs to see our class definition, we can skip the header file and place the definition right in the .C file.

The first thing we need to do is include a header file which will give us all we need to write an Expression. Since our plugin will live in isolation, we'll also add a `using` declaration to make life easier on ourselves; without it we'd have to declare the namespaces or prepend `sierra::Aria::` to lots of data types. So far, we have this:

```
#include <Aria_Plugin_Expression.h>
using namespace sierra::Aria;
```

Next we need the basic declaration of our class. This tells the compiler which methods and data members we want to have. Here's ours:

```
// Class definition -- could go in a header file.
class My_Density : public FAD_Expression
{
public:
  My_Density(Expression_Manager * const manager,
             const sierra::Identifier & expr_model_name,
             const Subdomain_Tag & subdomain_tag,
             const Int & subindex,
             const Phase_Label & phase_label,
             const sierra::String & params);
  virtual ~My_Density() {}
  virtual void compute_FAD_values(const ElementRange & elem_range);
private:
  Real a;
  Real b;
  Real c;
  Real d;

  Scalar_FAD_Expression_Handle values;
  Scalar_FAD_Expression_Handle temperature;
};
```

This is mostly boiler-plate code. Here we declare a constructor and an empty destructor and the method needed for computing our density's values. We also add some private data to store our polynomial coefficients.

Now it's time to write some code. First, we write the constructor which tells Aria, in a generic sense, what we provide (density), what we depend on (temperature) and what parameters we require from the user (a, b, c and d).

```
  My_Density::My_Density(Expression_Manager * const manager,
                         const sierra::Identifier & expr_model_name,
                         const Subdomain_Tag & subdomain_tag,
                         const Int & subindex,
                         const Phase_Label & phase_label,
                         const sierra::String & params) :
  FAD_Expression(manager,subdomain_tag,subindex,phase_label,params)
{
  my_tensor_order          = 0;
  my_expression_tag        = Expression_Tag(EXPR::DENSITY,NO_OP,subindex,phase_label);
  my_expression_model_name = expr_model_name;

  // List the expressions that are required for this model.
  const Expression_Tag temperature_tag(EXPR::TEMPERATURE,NO_OP,subindex,phase_label);

  add_prereq(temperature_tag,temperature);

  // Get my model parameters.
  a = b = c = d = 0.0;
  get_optional_param("A",a);
  get_optional_param("B",b);
  get_optional_param("C",c);
```

```
  get_optional_param("D",d);

  if(a == 0.0 && b == 0.0 && c == 0.0 && d == 0.0)
    {
      throw sierra::RuntimeUserError() << "ERROR: All MY_MODEL_DENSITY parameters are zero.";
    }

  // Make myself known to the manager.
  register_myself(values);
}
```

The `my_tensor_order` tells Aria what kind of field your Expression creates, viz. scalar, vector or tensor, tensor order 0, 1 or 2. If left unspecified, the default type is scalar (`my_tensor_order == 0`). Your expression also has a variable called `my_tensor_dimension` which is dimensionality of each tensor order. By default, `my_tensor_dimension` is set to the physical dimension of the problem, i.e., 2 for 2D and 3 for 3D. This variable is also set for scalar fields. Combined, these variables define the number of values required to fully specify your Expression at a point on the element: `pow(my_tensor_dimension,my_tensor_order)`. These also define the expected signature of the `values` data used below.

Next, we write the code that implements our density function. **Important note:** the `values` arrays are always initialized to zero before this method is called. Here `FAD_DoubleType` variables are used for non-constant values related to the `values` calculation.

```
void My_Density::compute_FAD_values(const ElementRange & elem_range)
{
for (auto elem : elem_range)
  {
  for(auto point : my_sizes.get_points())
    {
      const FAD_DoubleType & T = temperature(elem,point);
      values(elem,point)  = a + T*(b + T*(c + T*(d)));
    }
  }
}
```

Note that since our Expression is a scalar in this example, the signature of `values` is `values(elem,point)`. If our result was a vector or tensor, the signature would be `values(elem,point,r)` and `values(elem,point,r,s)`, respectively, where $0 \leq$ `r,s < my_tensor_dimension`.

The last thing we need to do is the actual plugin step. This one line of code,

```
ExprPluginFactory<My_Density> my_density_creator("MY_MODEL_DENSITY");
```

takes care of making your Expression known to Aria. The quoted string is the string that is used in your input file (except for the ending part, see section 18.3. If this string has any spaces in it, Aria will automatically replace them with an underscore.

For completeness the entire plugin code is given here.

```
#include <Aria_Plugin_Expression.h>

using namespace sierra::Aria;
```

```
// Class definition -- could go in a header file.
class My_Density : public FAD_Expression
{
public:
  My_Density(Expression_Manager * const manager,
     const sierra::Identifier & expr_model_name,
     const Subdomain_Tag & subdomain_tag,
     const Int & subindex,
     const Phase_Label & phase_label,
     const sierra::String & params);
  virtual ~My_Density() {}
  virtual void compute_FAD_values(const ElementRange & elem_range);
private:
  Real a;
  Real b;
  Real c;
  Real d;

  Scalar_FAD_Expression_Handle values;
  Scalar_FAD_Expression_Handle temperature;
};


My_Density::My_Density(Expression_Manager * const manager,
       const sierra::Identifier & expr_model_name,
       const Subdomain_Tag & subdomain_tag,
       const Int & subindex,
       const Phase_Label & phase_label,
       const sierra::String & params) :
  FAD_Expression(manager,subdomain_tag,subindex,phase_label,params)
{
  my_expression_tag    = Expression_Tag(EXPR::DENSITY,NO_OP,subindex,phase_label);
  my_expression_model_name = expr_model_name;

  // List the expressions that are required for this model.
  const Expression_Tag temperature_tag(EXPR::TEMPERATURE,NO_OP,subindex,phase_label);

  add_prereq(temperature_tag,temperature);

  // Get my model parameters.
  a = b = c = d = 0.0;
  get_optional_param("A",a);
  get_optional_param("B",b);
  get_optional_param("C",c);
  get_optional_param("D",d);

  if(a == 0.0 && b == 0.0 && c == 0.0 && d == 0.0)
    {
      throw sierra::RuntimeUserError() << "ERROR: All PLUGIN DENSITY parameters are zero.";
    }

  // Make myself known to the manager.
  register_myself(values);
}
```

```
void My_Density::compute_FAD_values(const ElementRange & elem_range)
{
for (auto elem : elem_range)
  {
    for (auto point : my_sizes.get_points())
    {
      const FAD_DoubleType & T = temperature(elem, point);
      values(elem, point)  = a + T*(b + T*(c + T*(d)));
    }
  }
}
ExprPluginFactory<My_Density> my_density_creator("MY_MODEL_DENSITY");
```

18.6 Testing Your Plugin

There are two good tests you can perform to test your plugin. The first is to run Aria in debug mode. To do this, add `-o dbg` to your `build` command line to build a debug executable. Then, add `-d` to your `sierra` command line to run the debug executable. In debug mode Aria will, among other things, perform bounds checking on your `FAD_MDArray` objects like `values(...)`, `dself(...)` and `sens(...)` in this example.

Secondly, if you hand-code your Newton sensitivities (not done in this example), you can test the coding of your sensitivities by adding `-arialog sens_check` to your `sierra` command line. This will cause Aria to compare the computed sensitivity values with numerical approximations. The sensitivity checker will cause your code to run slower but it will work in either debug or optimized mode. Aria's sensitivity checker is designed to only report discrepancies that have a high probability of being true errors so it's possible that it may miss some small errors. However, reported errors are most probably real.

Oftentimes the easiest way to debug your code is to print information to the screen. Aria provides a facility to support this. To print information to the log file, use the `arialog` C++ output stream. For example,

```
for (auto elem : elem_range)
{
  for(auto point : my_sizes.get_points())
  {
    ...
    values(elem,point) = ...
    arialog.m(LOG_PLUGIN) << "value(" << point << ") = " << value(elem,point) << endl;
    }
}
```

Then, you can activate this output by adding the option `-arialog plugin` to your `sierra` command line. If you leave off the `.m(LOG_PLUGIN)` part then it will always write your output to the log file. There's a performance penalty for having this code option present (even if out don't turn the logging on) so you probably want to remove it once you're done debugging.

Chapter 19

Dynamic Load Balancing

Aria supports dynamic load balancing of elements using the STK rebalance interface to Zoltan and Zoltan2. This is particularly useful for simulations with any of:

- Different equations solved on different mesh blocks (including thermal problems with chemistry source terms only on some blocks).
- Mesh adaptivity (either standard AMR or CDFEM)
- Multiple equation systems

By default when rebalance is enabled Aria will assign each element in the mesh a load based on the amount of time it takes to assemble its matrix contributions, and will use a multi-criteria rebalance in order to balance the load of each equation system simultaneously. The frequency that the load balance is checked can be controlled with the "REBALANCE TIME STEP FREQUENCY" and "MAXIMUM NUMBER OF LOAD BALANCES" line commands. Generally for problems without adaptivity performing a single rebalance after the first time step is sufficient, and for cases with significant differences in computational cost per element or multiple equation systems has been seen to provide a 50-100% speed-up in parallel in some cases.

Use of the dynamic load balancing capability has been tested with adaptive mesh refinement, CDFEM, contact, bulk nodes, pressurization models, and enclosure radiation.

Known Issue: Using rebalance with enclosure radiation requires that the viewfactors be recalculated every time the mesh is rebalanced, and is not currently compatible with reading viewfactors from a file.

See, also, the Zoltan homepage [31], the Zoltan User's Guide [32] and an overview of Zoltan [33].

19.1 ENABLE REBALANCE

`ENABLE REBALANCE [WITH THRESHOLD =` *`REAL`*`]`

Description — Enables load balancing for parallel simulations.

Summary — This command causes Aria to occasionally redistribute the mesh across the processors in a parallel run in order to (hopefully) balance the work. In order to perform this balancing, Aria must supply a "load" for each element on each processor to the Zoltan library. See the `REBALANCE LOAD MEASURE` command (19.2) for load measuring options.

The loads can be output to the results database by adding the following line to the `RESULTS OUTPUT` block in the input file:

```
Element Variables = rebalance->element_load as Load
```

The number optionally supplied as a threshold determines how far out of balance the load can become before load balancing is performed and defaults to 1.05.

Known Issue: Prior versions of Aria allowed the specification of a "Zoltan Parameters" block for rebalance. That has been deprecated and will not impact the simulation any more, however the parser will accept it for backwards compatibility.

Parent Block(s) `ARIA_REGION`

19.2 REBALANCE LOAD MEASURE

`REBALANCE LOAD MEASURE = STRING`

Description Selects the method for measuring element loads for rebalancing.

Summary Valid options are `ELEMENT` (default), `PROCESSOR` and `CONSTANT`.

`ELEMENT` : (default) assigns an element weight equal to the total cost of assembling the element for a timestep divided by the number of nonlinear iterations.

`PROCESSOR` : assigns same weight to all elements on a processor equal to the average cost of assembling an element for a timestep divided by the number of nonlinear iterations.

`CONSTANT` : assigns the same weight to each element, and is useful for regression testing.

Parent Block(s) `ARIA_REGION`

19.3 MAXIMUM NUMBER OF REBALANCES

`MAXIMUM NUMBER OF REBALANCES = INT`

Description Disables dynamics mesh rebalancing after the specified number of mesh rebalances have been performed.

Summary The number of rebalances only increments if elements are actually migrated across processors. By default, Aria will rebalance as many times as needed.

Parent Block(s) `ARIA_REGION`

19.4 REBALANCE TIME STEP FREQUENCY

`REBALANCE TIME STEP FREQUENCY = INT`

Description Causes Aria to only examine the load balance every N steps, where N is the number supplied by this command line.

Summary By default, Aria will check the load balance every time step.

Parent Block(s) `ARIA_REGION`

Chapter 20

Linear Solver Reference

20.1 Overview

For most problems Aria makes use of the Trilinos and Aztec equation solvers. Another solver GDSW, geared specifically toward incompressible flows is also available. These solvers provide an object-oriented interface to allow for flexible construction matrix and vector algorithms. Within Aria one provides the solver parameters at the domain scope and then references the solver within the Aria region. See, also, the Trilinos parameters online reference.

Within the input file linear solver command block will appear at the Domain scope as illustrated below. Note that different linear solvers can be used for different equation sets. Furthermore a solver command block can appear in the input but not be used.

```
Begin Sierra
  .
  Begin Trilinos Equation Solver first_solver
    .
    Trilinos linear solver commands
    .
  End
  .
  .
  Begin Aztec Equation Solver second_solver
    .
    Aztec linear solver commands
    .
  End
  .
  Begin Aztec Equation Solver third_solver
    .
    Aztec linear solver commands
    .
  End
  .
  Begin Procedure My_Aria_Procedure
     .
     Begin Aria Region My_Region
      .
      use linear solver first_solver
      .
     End
```

```
    Begin Aria Region My_Other_Region
     .
     use linear solver third_solver
     .
    End

  End
  .
End Sierra
```

20.2 Ifpack2 Equation Solver

Scope: Sierra

Begin Ifpack2 Equation Solver *Solver Name*

Bc Enforcement {=|are|is} *bc enforcement*

Convergence Tolerance {=|are|is} *convergence tolerance*

Explicit Residual Scaling {=|are|is} *explicit residual scaling*

Implicit Residual Scaling {=|are|is} *implicit residual scaling*

Maximum Iterations {=|are|is} *maximum iterations*

Maximum Restarts {=|are|is} *maximum restarts*

Num Blocks {=|are|is} *num blocks*

Solution Method {=|are|is} *solution method*

Chebyshev: Degree {=|are|is} *chebyshev: degree*

Fact: Absolute Threshold {=|are|is} *fact: absolute threshold*

Fact: Drop Tolerance {=|are|is} *fact: drop tolerance*

Fact: Iluk Level-Of-Fill {=|are|is} *fact: iluk level-of-fill*

Fact: Ilut Level-Of-Fill {=|are|is} *fact: ilut level-of-fill*

Fact: Relative Threshold {=|are|is} *fact: relative threshold*

Fact: Relax Value {=|are|is} *fact: relax value*

Preconditioner Type {=|are|is} *preconditioner type*

Relaxation: Backward Mode {=|are|is} *relaxation: backward mode*

Relaxation: Check Diagonal Entries {=|are|is} *relaxation: check diagonal entries*

Relaxation: Damping Factor {=|are|is} *relaxation: damping factor*

Relaxation: Fix Tiny Diagonal Entries {=|are|is} *relaxation: fix tiny diagonal entries*

Relaxation: L1 Eta {=|are|is} *relaxation: l1 eta*

Relaxation: Min Diagonal Value {=|are|is} *relaxation: min diagonal value*

Relaxation: Sweeps {=|are|is} *relaxation: sweeps*

Relaxation: Type {=|are|is} *relaxation: type*

Relaxation: Use L1 {=|are|is} *relaxation: use l1*

Relaxation: Zero Starting Solution {=|are|is} *relaxation: zero starting solution*

```
    Schwarz:  Combine Mode {=|are|is} schwarz:  combine mode
    Schwarz:  Filter Singletons {=|are|is} schwarz:  filter singletons
    Schwarz:  Inner Preconditioner Name {=|are|is} schwarz:  inner preconditioner name

    Schwarz:  Num Iterations {=|are|is} schwarz:  num iterations
    Schwarz:  Overlap Level {=|are|is} schwarz:  overlap level
    Schwarz:  Use Reordering {=|are|is} schwarz:  use reordering
    Schwarz:  Zero Starting Solution {=|are|is} schwarz:  zero starting solution
    Begin Schwarz Inner Preconditioner Parameters Inner Preconditioner Solver Name
    End
End
```

Summary Solver parameters for the Tpetra based equation solver.

Description Block for exercising tpetra-based Ifpack2 preconditioners underneath the Belos solver library.

20.2.1 Bc Enforcement

Scope: Ifpack2 Equation Solver

Summary Controls the way Dirichlet BCs are enforced.

Description Valid values for this line-command are contained in the Tpetra_bc_enforcement enum.

20.2.2 Convergence Tolerance

Scope: Ifpack2 Equation Solver

```
Convergence Tolerance {=|are|is} convergence tolerance
```

Parameter	Value	Default
convergence tolerance	`real`	`undefined`

Summary Relative residual convergence tolerance.

20.2.3 Explicit Residual Scaling

Scope: Ifpack2 Equation Solver

Summary Residual norm scaling method

Description Residual norm scaling method

20.2.4 Implicit Residual Scaling

Scope: Ifpack2 Equation Solver

Summary Residual norm scaling method

Description Residual norm scaling method

20.2.5 Maximum Iterations

Scope: Ifpack2 Equation Solver

`Maximum Iterations {=|are|is}` *`maximum iterations`*

Parameter	Value	Default
maximum iterations	`integer`	`undefined`

Summary Maximum number of solution method iterations.

Description This is Maximum Iterations

20.2.6 Maximum Restarts

Scope: Ifpack2 Equation Solver

`Maximum Restarts {=|are|is}` *`maximum restarts`*

Parameter	Value	Default
maximum restarts	`integer`	`undefined`

Summary Number of solution method restart iterations.

Description This is Restart Iterations

20.2.7 Num Blocks

Scope: Ifpack2 Equation Solver

`Num Blocks {=|are|is}` *`num blocks`*

Parameter	Value	Default
num blocks	`integer`	`undefined`

Summary The number of vectors (or blocks, in the case of multiple right-hand sides) allocated for the Krylov basis. Number of solution method restart iterations.

Description This is Restart Length

20.2.8 Solution Method

Scope: Ifpack2 Equation Solver

Summary Selection of the linear-system solution algorithm.

Description The Tpetra_solver_methods enum contains valid values for this line-command.

20.2.9 Chebyshev: Degree

Scope: Ifpack2 Equation Solver

`Chebyshev: Degree {=|are|is}` *`chebyshev: degree`*

Parameter	Value	Default
chebyshev: degree	`integer`	`undefined`

Summary chebyshev: degree

Description Degree of the Chebyshev polynomial

20.2.10 Fact: Absolute Threshold

Scope: Ifpack2 Equation Solver

`Fact: Absolute Threshold {=|are|is}` *`fact: absolute threshold`*

Parameter	Value	Default
fact: absolute threshold	`real`	`undefined`

Summary fact: absolute threshold

Description Prior to the factorization, each diagonal entry is updated by adding this value (with the sign of the actual diagonal entry). Can be combined with "fact: relative threshold". The matrix remains unchanged.

20.2.11 Fact: Drop Tolerance

Scope: Ifpack2 Equation Solver

`Fact: Drop Tolerance {=|are|is}` *`fact: drop tolerance`*

Parameter	Value	Default
fact: drop tolerance	`real`	`undefined`

Summary fact: drop tolerance

Description A threshold for dropping entries.

20.2.12 Fact: Iluk Level-Of-Fill

Scope: Ifpack2 Equation Solver

`Fact: Iluk Level-Of-Fill {=|are|is}` *`fact: iluk level-of-fill`*

Parameter	Value	Default
fact: iluk level-of-fill	`real`	`undefined`

Summary fact: iluk level-of-fill

Description Level-of-fill of the factorization.

20.2.13 Fact: Ilut Level-Of-Fill

Scope: Ifpack2 Equation Solver

`Fact: Ilut Level-Of-Fill {=|are|is}` *`fact: ilut level-of-fill`*

Parameter	Value	Default
fact: ilut level-of-fill	`real`	`undefined`

Summary fact: ilut level-of-fill

Description Maximum number of entries to keep in each row of L and U. Each row of L (U) will have at most âŇĹ(level-of-fillâĹŠ1)nnz(A)âŇĽ 2n nonzero entries, where nnz(A) is the number of nonzero entries in the matrix, and n is the number of rows. ILUT always keeps the diagonal entry in the current row, regardless of the drop tolerance or fill level. Note: This is different from the p in the classic algorithm in [13].

20.2.14 Fact: Relative Threshold

Scope: Ifpack2 Equation Solver

`Fact: Relative Threshold {=|are|is}` *`fact: relative threshold`*

Parameter	Value	Default
fact: relative threshold	`real`	`undefined`

Summary fact: relative threshold

Description Prior to the factorization, each diagonal element is scaled by this factor (not including contribution specified by "fact: absolute threshold"). Can be combined with "fact: absolute threshold". The matrix remains unchanged.

20.2.15 Fact: Relax Value

Scope: Ifpack2 Equation Solver

`Fact: Relax Value {=|are|is}` *`fact: relax value`*

Parameter	Value	Default
fact: relax value	`real`	`undefined`

Summary fact: relax value

Description MILU diagonal compensation value. Entries dropped during factorization times this factor are added to diagonal entries.

20.2.16 Preconditioner Type

Scope: Ifpack2 Equation Solver

Summary preconditioner type

Description Type of preconditioner used for solve

20.2.17 Relaxation: Backward Mode

Scope: Ifpack2 Equation Solver

Summary relaxation: backward mode

Description Governs whether Gauss-Seidel is done in forward-mode (false) or backward-mode (true). Only valid for "Gauss-Seidel" type.

20.2.18 Relaxation: Check Diagonal Entries

Scope: Ifpack2 Equation Solver

Summary relaxation: check diagonal entries

Description If true, the compute() method will do extra work (computation and communication) to count diagonal entries that are zero, have negative real part, or are small in magnitude. This information can be later shown in the description.

20.2.19 Relaxation: Damping Factor

Scope: Ifpack2 Equation Solver

`Relaxation: Damping Factor {=|are|is}` *`relaxation: damping factor`*

Parameter	Value	Default
relaxation: damping factor	`real`	`undefined`

Summary relaxation: damping factor

Description The value of the damping factor omega for the relaxation.

20.2.20 Relaxation: Fix Tiny Diagonal Entries

Scope: Ifpack2 Equation Solver

Summary relaxation: fix tiny diagonal entries

Description If true, the compute() method will do extra work (computation only, no MPI communication) to fix diagonal entries. Specifically, the diagonal val- ues with a magnitude smaller than the magnitude of the threshold relaxation: min diagonal value are increased to threshold for the diagonal inversion. The matrix is not modified, instead the updated diag- onal values are stored. If the threshold is zero, only the diagonal entries that are exactly zero are replaced with a small nonzero value (machine precision).

20.2.21 Relaxation: L1 Eta

Scope: Ifpack2 Equation Solver

`Relaxation: L1 Eta {=|are|is}` *`relaxation: l1 eta`*

Parameter	Value	Default
relaxation: l1 eta	`real`	`undefined`

Summary relaxation: l1 eta

Description Magnitude of eta parameter for l1 variant of Gauss-Seidel. Only used if "relaxation: use l1" is true.

20.2.22 Relaxation: Min Diagonal Value

Scope: Ifpack2 Equation Solver

`Relaxation: Min Diagonal Value {=|are|is}` *`relaxation: min diagonal value`*

Parameter	Value	Default
relaxation: min diagonal value	`real`	`undefined`

Summary relaxation: min diagonal value

Description The threshold value used in "relaxation: fix tiny diagonal entries". Only used if "relaxation: fix tiny diagonal entries" is true.

20.2.23 Relaxation: Sweeps

Scope: Ifpack2 Equation Solver

`Relaxation: Sweeps {=|are|is}` *`relaxation: sweeps`*

Parameter	Value	Default
relaxation: sweeps	`integer`	`undefined`

Summary relaxation: sweeps

Description Number of sweeps of the relaxation.

20.2.24 Relaxation: Type

Scope: Ifpack2 Equation Solver

Summary relaxation: type

Description Type of point relaxation scheme for preconditioning

20.2.25 Relaxation: Use L1

Scope: Ifpack2 Equation Solver

Summary relaxation: use l1

Description Use the l1 variant of Jacobi or Gauss-Seidel.

20.2.26 Relaxation: Zero Starting Solution

Scope: Ifpack2 Equation Solver

Summary relaxation: zero starting solution

Description Governs whether or not Ifpack2 uses existing values in the left hand side vector. If true, Ifpack2 will fill it with zeros before applying relaxation sweeps which may make the first sweeep more efficient.

20.2.27 Schwarz: Combine Mode

Scope: Ifpack2 Equation Solver

Summary schwarz: combine mode

Description The rule for combining incoming data with existing data in overlap regions. Accepted values: ADD, ZERO, INSERT, REPLACE, ABSMAX

20.2.28 Schwarz: Filter Singletons

Scope: Ifpack2 Equation Solver

Summary schwarz: filter singletons

Description If true, exclude rows with just a single entry on the calling process.

20.2.29 Schwarz: Inner Preconditioner Name

Scope: Ifpack2 Equation Solver

Summary schwarz: inner preconditioner name

Description Type of preconditioner used for the inner solve in a schwarz domain decomposition method

20.2.30 Schwarz: Num Iterations

Scope: Ifpack2 Equation Solver

`Schwarz: Num Iterations {=|are|is}` *`schwarz: num iterations`*

Parameter	Value	Default
schwarz: num iterations	integer	undefined

Summary schwarz: num iterations

Description Number of iterations to perform.

20.2.31 Schwarz: Overlap Level

Scope: Ifpack2 Equation Solver

`Schwarz: Overlap Level {=|are|is}` *`schwarz: overlap level`*

Parameter	Value	Default
schwarz: overlap level	integer	undefined

Summary schwarz: overlap level

Description The level of overlap (0 corresponds to no overlap).

20.2.32 Schwarz: Use Reordering

Scope: Ifpack2 Equation Solver

Summary schwarz: use reordering

Description If true, local matrix is reordered before computing subdomain solver.

20.2.33 Schwarz: Zero Starting Solution

Scope: Ifpack2 Equation Solver

Summary schwarz: zero starting solution

Description Governs whether or not Ifpack2 uses existing values in the left hand side vector. If true, Ifpack2 will fill it with zeros before applying relaxation sweeps which may make the first sweeep more efficient.

20.3 Trilinos Equation Solver

Scope: Sierra

```
Begin Trilinos Equation Solver SolverName
    Bc Enforcement {=|are|is} BcEnforcement
    Debug Output Level {=|are|is} Level
    Debug Output Path {=|are|is} DebugOutput
    Determine Sharing {=|are|is} FeiDetermineSharing
    Fei Error Behavior {=|are|is} FeiErrorBehavior
    Fei Output Level {=|are|is} FeiOutputLevels
    Ilu Fill {=|are|is} ilu_fill_level
    Ilu Threshold {=|are|is} Threashold
    Matrix Format {=|are|is} MatrixFormat
    Matrix Reduction {=|are|is} AztecReductionType
    Matrix Scaling {=|are|is} MatrixScaling
    Matrix Viewer {=|are|is} Machine:port
    Maximum Iterations {=|are|is} max_iterations
    Num Levels {=|are|is} Num_levels
    Param-Bool Parameter_name Value Bool
    Param-Int Parameter_name Value integer_value
    Param-Real Parameter_name Value Real_value
    Param-String Parameter_name Value String_value
    Polynomial Order {=|are|is} polynomial_order
    Preconditioner Subdomain Overlap {=|are|is} Overlap
    Preconditioning Method {=|are|is} TrilinosPrecondMethods
    Preconditioning Steps {=|are|is} preconditioning_steps
    Residual Norm Scaling {=|are|is} AztecResidualNormScaling
    Residual Norm Tolerance {=|are|is} residual_norm_tolerance
    Restart Iterations {=|are|is} restart_iterations
    Row Ordering {=|are|is} RowOrdering
    Shared Ownership Rule {=|are|is} SharedOwnershipRule
    Solution Method {=|are|is} TrilinosSolverMethods
    Solve Transpose {=|are|is} Option
    Select Fei {=|are|is} WhichFEI
    Begin Teuchos Parameter Block Teuchos Parameter Block Name
```

```
    End
End
```

Summary A set of solver parameters for Trilinos equation solver.

20.3.1 Bc Enforcement

Scope: Trilinos Equation Solver

`Bc Enforcement {= | are | is}` *`BcEnforcement`*

Parameter	Value	Default
BcEnforcement	`{exact | exact_no_column_mod | remove | solver | solver_no_column_mod}`	`undefined`

Summary Controls the way Dirichlet BCs are enforced.

Description Valid values for this line-command are contained in the BcEnforcement enum.

20.3.2 Debug Output Level

Scope: Trilinos Equation Solver

`Debug Output Level {= | are | is}` *`Level`*

Parameter	Value	Default
Level	`integer`	`undefined`

Summary Output level for debugging. Generally 0 means no solver screen output, and higher values of this parameter correspond to more screen output.

20.3.3 Debug Output Path

Scope: Trilinos Equation Solver

`Debug Output Path {= | are | is}` *`DebugOutput`*

Parameter	Value	Default
DebugOutput	`string`	`undefined`

Summary Specify path where debug-logs and other debug output files will be placed.

20.3.4 Determine Sharing

Scope: Trilinos Equation Solver

`Determine Sharing {= | are | is}` *`FeiDetermineSharing`*

Parameter	Value	Default
FeiDetermineSharing	{fei | sierra}	undefined

Summary Whether FEI should determine sharing internally or receive the info from Sierra.

Description By default the Eqns::LinearSystem class tells the fei layer which nodes are shared and which processors share them. If this option is set to on, then the fei layer internally determines the sharing info. Requires the fei layer to perform extra communication during the intialize phase.

20.3.5 Fei Error Behavior

Scope: Trilinos Equation Solver

`Fei Error Behavior {= | are | is}` *FeiErrorBehavior*

Parameter	Value	Default
FeiErrorBehavior	{abort | returncode}	undefined

Summary Set FEI error behavior to abort (rather than the default which is to simply print a message and return an integer error code).

Description This is becoming less relevant starting with FEI version 2.11, as the FEI begins to adopt exception handling and abandon the antiquated int-error-code interfaces.

20.3.6 Fei Output Level

Scope: Trilinos Equation Solver

`Fei Output Level {= | are | is}` *FeiOutputLevels*

Parameter	Value	Default
FeiOutputLevels	{all | brief_logs | full_logs | matrix_files | none | stats}	undefined

Summary Control the amount of output produced by FEI.

Description Output level controls the amount of debugging information (screen output, matrix/vector files and log files) produced by FEI. The FeiOutputLevels enum contains the valid values for this line-command. The location of files can be controlled by the 'debug output path' line-command.

20.3.7 Ilu Fill

Scope: Trilinos Equation Solver

`Ilu Fill {= | are | is}` *ilu_fill_level*

Parameter	Value	Default
ilu_fill_level	integer	undefined

Summary Fill-in parameter for incomplete factorizations.

20.3.8 Ilu Threshold

Scope: Trilinos Equation Solver

`Ilu Threshold {= | are | is}` *`Threashold`*

Parameter	Value	Default
Threashold	`real`	`undefined`

Summary Threshold parameter for incomplete factorizations.

20.3.9 Matrix Format

Scope: Trilinos Equation Solver

`Matrix Format {= | are | is}` *`MatrixFormat`*

Parameter	Value	Default
MatrixFormat	`{csr | msr | vbr}`	`undefined`

Summary Storage format for the matrix.

Description This parameter only applies to Trilinos and Aztec.

20.3.10 Matrix Reduction

Scope: Trilinos Equation Solver

`Matrix Reduction {= | are | is}` *`AztecReductionType`*

Parameter	Value	Default
AztecReductionType	`{fei-remove-slaves}`	`undefined`

Summary Remove constraint equations from matrix.

Description Dependent degrees of freedom in constraints are projected out of the equation space, yielding a linear system with smaller dimension, and retaining positive definiteness (whereas the alternative, an augmented matrix arising from the lagrange multiplier formulation, is indefinite).

20.3.11 Matrix Scaling

Scope: Trilinos Equation Solver

`Matrix Scaling {= | are | is}` *`MatrixScaling`*

Parameter	Value	Default
MatrixScaling	`{block-jacobi | jacobi | none | row-sum | symmetric-diagonal | symmetric-row-sum | user supplied}`	`undefined`

Summary Scaling to be applied to the matrix.

Description Matrix scaling is a pre-solve operation, typically modifying the matrix in place, whereas preconditioning is performed at each iteration during the solve and doesn't modify the actual matrix.

20.3.12 Matrix Viewer

Scope: Trilinos Equation Solver

`Matrix Viewer {=|are|is}` *`Machine:port`*

Parameter	Value	Default
Machine:port	`string`	`undefined`

Summary Host and port-number where matvis is running.

20.3.13 Maximum Iterations

Scope: Trilinos Equation Solver

`Maximum Iterations {=|are|is}` *`max_iterations`*

Parameter	Value	Default
max_iterations	`integer`	`undefined`

Summary Maximum number of solution method iterations.

20.3.14 Num Levels

Scope: Trilinos Equation Solver

`Num Levels {=|are|is}` *`Num_levels`*

Parameter	Value	Default
Num_levels	`integer`	`undefined`

Summary Number of levels for multi-level/multi-grid solvers.

20.3.15 Param-Bool

Scope: Trilinos Equation Solver

`Param-Bool` *`Parameter_name`* `Value` *`Bool`*

Parameter	Value	Default
Parameter_name	`"string"`	`undefined`
Bool	`{false|true}`	`undefined`

Summary String-Key/Boolean-Value pair to be passed to solver.

Description Syntax: 'PARAM-BOOL "some-name" VALUE true | false'

20.3.16 Param-Int

Scope: Trilinos Equation Solver

`Param-Int` *`Parameter_name`* `Value` *`integer_value`*

Parameter	Value	Default
Parameter_name	`"string"`	`undefined`
integer_value	`integer`	`undefined`

Summary String-Key/Integer-Value pair to be passed to solver.

Description Syntax: 'PARAM-INT "some-name" VALUE 123' Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system. Can also be used to pass parameters to Aztec, e.g.:

PARAM-REAL "AZ_ilut_fill" VALUE 3

20.3.17 Param-Real

Scope: Trilinos Equation Solver

`Param-Real` *`Parameter_name`* `Value` *`Real_value`*

Parameter	Value	Default
Parameter_name	`"string"`	`undefined`
Real_value	`real`	`undefined`

Summary String-Key/Real-Value pair to be passed to solver.

Description Syntax: 'PARAM-REAL "some-name" VALUE 1.23' Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system. Can also be used to pass parameters to Aztec, e.g.:

PARAM-REAL "AZ_drop" VALUE 1.e-8

20.3.18 Param-String

Scope: Trilinos Equation Solver

`Param-String` *`Parameter_name`* `Value` *`String_value`*

Parameter	Value	Default
Parameter_name	`"string"`	`undefined`
String_value	`"string"`	`undefined`

Summary Key/Value string-pair to be passed to solver.

Description Syntax: 'PARAM-STRING "some-name" VALUE "some-value"' (Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system.)

20.3.19 Polynomial Order

Scope: Trilinos Equation Solver

`Polynomial Order {= | are | is}` *`polynomial_order`*

Parameter	Value	Default
polynomial_order	integer	undefined

Summary Polynomial order of preconditioning method.

20.3.20 Preconditioner Subdomain Overlap

Scope: Trilinos Equation Solver

`Preconditioner Subdomain Overlap {= | are | is}` *`Overlap`*

Parameter	Value	Default
Overlap	integer	undefined

Summary Ovrlap of Schwarz subdomains (eg, 0,1 or 2).

20.3.21 Preconditioning Method

Scope: Trilinos Equation Solver

`Preconditioning Method {= | are | is}` *`TrilinosPrecondMethods`*

Parameter	Value	Default
TrilinosPrecondMethods	{dd-bilu | dd-icc | dd-ilu | dd-ilut | dd-lu | dd-rilu | jacobi | least-squares | multilevel | neumann | none}	undefined

---

Summary Selection of the preconditioning method.

Description Valid values for this are contained in the TrilinosPrecondMethods enum.

20.3.22 Preconditioning Steps

Scope: Trilinos Equation Solver

`Preconditioning Steps {= | are | is}` *`preconditioning_steps`*

Parameter	Value	Default
preconditioning_steps	integer	undefined

Summary Number of Jacobi, Gauss-Seidel, or other preconditioning methods' applications per iteration.

20.3.23 Residual Norm Scaling

Scope: Trilinos Equation Solver

`Residual Norm Scaling {=|are|is}` *`AztecResidualNormScaling`*

Parameter	Value	Default
AztecResidualNormScaling	`{anorm|none|r0|rhs}`	`undefined`

Summary Scaling method for the residual norm.

Description Determines the residual expression used in convergence checks and printing.

20.3.24 Residual Norm Tolerance

Scope: Trilinos Equation Solver

`Residual Norm Tolerance {=|are|is}` *`residual_norm_tolerance`*

Parameter	Value	Default
residual_norm_tolerance	`real`	`undefined`

Summary Iterative solution method relative residual convergence tolerance.

20.3.25 Restart Iterations

Scope: Trilinos Equation Solver

`Restart Iterations {=|are|is}` *`restart_iterations`*

Parameter	Value	Default
restart_iterations	`integer`	`undefined`

Summary Number of iterations between GMRES restarts.

20.3.26 Row Ordering

Scope: Trilinos Equation Solver

`Row Ordering {=|are|is}` *`RowOrdering`*

Parameter	Value	Default
RowOrdering	`{rows_with_local_cols_first}`	`undefined`

Summary Specify that rows with only local columns should come first in the ordering.

20.3.27 Shared Ownership Rule

Scope: Trilinos Equation Solver

`Shared Ownership Rule {= | are | is}` *SharedOwnershipRule*

Parameter	Value	Default
SharedOwnershipRule	{low-numbered-proc | proc-with-local-elem | sierra_specifies}	undefined

Summary Controls the way owning processors are chosen for shared nodes in the FEI.

Description 'low-numbered-proc' is the default.

'proc-with-local-elem' is another valid value.

'sierra_specifies' is the other valid value

20.3.28 Solution Method

Scope: Trilinos Equation Solver

`Solution Method {= | are | is}` *TrilinosSolverMethods*

Parameter	Value	Default
TrilinosSolverMethods	{amesos-klu | amesos-superlu | amesos-umfpack | bicgstab | cg | cgs | gmres | lu | tfqmr}	undefined

Summary Selection of the linear-system solution algorithm.

Description The TrilinosSolverMethods enum contains valid values for this line-command. There are several iterative methods (Krylov subspace algorithms), as well as some sparse direct methods available through the Amesos interface package. At this time (Oct'05) not all of the listed direct solvers are available. See the specific entries in the TrilinosSolverMethods enum below for more information.

20.3.29 Solve Transpose

Scope: Trilinos Equation Solver

`Solve Transpose {= | are | is}` *Option*

Parameter	Value	Default
Option	string	undefined

Summary Whether to solve for transpose of system matrix.

20.3.30 Select Fei

Scope: Trilinos Equation Solver

`Select Fei {= | are | is}` *WhichFEI*

Parameter	Value	Default
WhichFEI	{new | old}	undefined

Summary Selection of old vs new fei.

Description This parameter will be deprecated, it is used as a transition aid during some FEI refactoring.

20.4 Aztec Equation Solver

Scope: Sierra

```
Begin Aztec Equation Solver Solver Name
    Bc Enforcement {=|are|is} BcEnforcement
    Debug Output Level {=|are|is} Level
    Debug Output Path {=|are|is} DebugOutput
    Determine Sharing {=|are|is} FeiDetermineSharing
    Fei Error Behavior {=|are|is} FeiErrorBehavior
    Fei Output Level {=|are|is} FeiOutputLevels
    Ilu Fill {=|are|is} ilu_fill_level
    Ilu Omega {=|are|is} Value
    Ilu Threshold {=|are|is} Threashold
    Matrix Format {=|are|is} MatrixFormat
    Matrix Reduction {=|are|is} AztecReductionType
    Matrix Scaling {=|are|is} MatrixScaling
    Matrix Viewer {=|are|is} Machine:port
    Maximum Iterations {=|are|is} max_iterations
    Num Levels {=|are|is} Num_levels
    Orthog Method {=|are|is} OrthogMethod
    Param-Int Parameter_name Value integer_value
    Param-Real Parameter_name Value Real_value
    Param-String Parameter_name Value String_value
    Polynomial Order {=|are|is} polynomial_order
    Preconditioner Subdomain Overlap {=|are|is} Overlap
    Preconditioning Method {=|are|is} AztecPrecondMethods
    Preconditioning Steps {=|are|is} preconditioning_steps
    Residual Norm Scaling {=|are|is} AztecResidualNormScaling
    Residual Norm Tolerance {=|are|is} residual_norm_tolerance
    Restart Iterations {=|are|is} restart_iterations
    Shared Ownership Rule {=|are|is} SharedOwnershipRule
    Solution Method {=|are|is} AztecSolverMethods
    Select Fei {=|are|is} WhichFEI
End
```

Summary A set of solver parameters for Aztec equation solver.

20.4.1 Bc Enforcement

Scope: Aztec Equation Solver

`Bc Enforcement {= | are | is}` *`BcEnforcement`*

Parameter	Value	Default
BcEnforcement	`{exact | exact_no_column_mod | remove | solver | solver_no_column_mod}`	`undefined`

Summary Controls the way Dirichlet BCs are enforced.

Description Valid values for this line-command are contained in the BcEnforcement enum.

20.4.2 Debug Output Level

Scope: Aztec Equation Solver

`Debug Output Level {= | are | is}` *`Level`*

Parameter	Value	Default
Level	`integer`	`undefined`

Summary Output level for debugging. Generally 0 means no solver screen output, and higher values of this parameter correspond to more screen output.

20.4.3 Debug Output Path

Scope: Aztec Equation Solver

`Debug Output Path {= | are | is}` *`DebugOutput`*

Parameter	Value	Default
DebugOutput	`string`	`undefined`

Summary Specify path where debug-logs and other debug output files will be placed.

20.4.4 Determine Sharing

Scope: Aztec Equation Solver

`Determine Sharing {= | are | is}` *`FeiDetermineSharing`*

Parameter	Value	Default
FeiDetermineSharing	`{fei | sierra}`	`undefined`

Summary Whether FEI should determine sharing internally or receive the info from Sierra.

Description By default the Eqns::LinearSystem class tells the fei layer which nodes are shared and which processors share them. If this option is set to on, then the fei layer internally determines the sharing info. Requires the fei layer to perform extra communication during the intialize phase.

20.4.5 Fei Error Behavior

Scope: Aztec Equation Solver

`Fei Error Behavior {=|are|is}` *`FeiErrorBehavior`*

Parameter	Value	Default
FeiErrorBehavior	`{abort|returncode}`	`undefined`

Summary — Set FEI error behavior to abort (rather than the default which is to simply print a message and return an integer error code).

Description — This is becoming less relevant starting with FEI version 2.11, as the FEI begins to adopt exception handling and abandon the antiquated int-error-code interfaces.

20.4.6 Fei Output Level

Scope: Aztec Equation Solver

`Fei Output Level {=|are|is}` *`FeiOutputLevels`*

Parameter	Value	Default
FeiOutputLevels	`{all|brief_logs|full_logs|matrix_files|none|stats}`	`undefined`

Summary — Control the amount of output produced by FEI.

Description — Output level controls the amount of debugging information (screen output, matrix/vector files and log files) produced by FEI. The FeiOutputLevels enum contains the valid values for this line-command. The location of files can be controlled by the 'debug output path' line-command.

20.4.7 Ilu Fill

Scope: Aztec Equation Solver

`Ilu Fill {=|are|is}` *`ilu_fill_level`*

Parameter	Value	Default
ilu_fill_level	`integer`	`undefined`

Summary — Fill-in parameter for incomplete factorizations.

20.4.8 Ilu Omega

Scope: Aztec Equation Solver

`Ilu Omega {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Omega parameter, dd-rilu uses ILU(k,w), w==omega

20.4.9 Ilu Threshold

Scope: Aztec Equation Solver

`Ilu Threshold {= | are | is}` *`Threashold`*

Parameter	Value	Default
Threashold	`real`	`undefined`

Summary Threshold parameter for incomplete factorizations.

20.4.10 Matrix Format

Scope: Aztec Equation Solver

`Matrix Format {= | are | is}` *`MatrixFormat`*

Parameter	Value	Default
MatrixFormat	`{csr	msr	vbr}`	`undefined`

Summary Storage format for the matrix.

Description This parameter only applies to Trilinos and Aztec.

20.4.11 Matrix Reduction

Scope: Aztec Equation Solver

`Matrix Reduction {= | are | is}` *`AztecReductionType`*

Parameter	Value	Default
AztecReductionType	`{fei-remove-slaves}`	`undefined`

Summary Remove constraint equations from matrix.

Description Dependent degrees of freedom in constraints are projected out of the equation space, yielding a linear system with smaller dimension, and retaining positive definiteness (whereas the alternative, an augmented matrix arising from the lagrange multiplier formulation, is indefinite).

20.4.12 Matrix Scaling

Scope: Aztec Equation Solver

`Matrix Scaling {= | are | is}` *`MatrixScaling`*

Parameter	Value	Default
MatrixScaling	`{block-jacobi	jacobi	none	row-sum	symmetric-diagonal	symmetric-row-sum	user supplied}`	`undefined`

Summary Scaling to be applied to the matrix.

Description Matrix scaling is a pre-solve operation, typically modifying the matrix in place, whereas preconditioning is performed at each iteration during the solve and doesn't modify the actual matrix.

20.4.13 Matrix Viewer

Scope: Aztec Equation Solver

`Matrix Viewer {= | are | is}` *`Machine:port`*

Parameter	Value	Default
Machine:port	`string`	`undefined`

Summary Host and port-number where matvis is running.

20.4.14 Maximum Iterations

Scope: Aztec Equation Solver

`Maximum Iterations {= | are | is}` *`max_iterations`*

Parameter	Value	Default
max_iterations	`integer`	`undefined`

Summary Maximum number of solution method iterations.

20.4.15 Num Levels

Scope: Aztec Equation Solver

`Num Levels {= | are | is}` *`Num_levels`*

Parameter	Value	Default
Num_levels	`integer`	`undefined`

Summary Number of levels for multi-level/multi-grid solvers.

20.4.16 Orthog Method

Scope: Aztec Equation Solver

`Orthog Method {= | are | is}` *`OrthogMethod`*

Parameter	Value	Default
OrthogMethod	`{classical	modified}`	`undefined`

Summary User can choose orthogonalization method used by Aztec GMRES.

Description Valid values for this line-command are contained in the OrthogMethod enum.

20.4.17 Param-Int

Scope: Aztec Equation Solver

`Param-Int` *`Parameter_name`* `Value` *`integer_value`*

Parameter	Value	Default
Parameter_name	"string"	undefined
integer_value	integer	undefined

Summary String-Key/Integer-Value pair to be passed to solver.

Description Syntax: 'PARAM-INT "some-name" VALUE 123' Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system. Can also be used to pass parameters to Aztec, e.g.:

PARAM-REAL "AZ_ilut_fill" VALUE 3

20.4.18 Param-Real

Scope: Aztec Equation Solver

`Param-Real` *`Parameter_name`* `Value` *`Real_value`*

Parameter	Value	Default
Parameter_name	"string"	undefined
Real_value	real	undefined

Summary String-Key/Real-Value pair to be passed to solver.

Description Syntax: 'PARAM-REAL "some-name" VALUE 1.23' Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system. Can also be used to pass parameters to Aztec, e.g.:

PARAM-REAL "AZ_drop" VALUE 1.e-8

20.4.19 Param-String

Scope: Aztec Equation Solver

`Param-String` *`Parameter_name`* `Value` *`String_value`*

Parameter	Value	Default
Parameter_name	"string"	undefined
String_value	"string"	undefined

Summary Key/Value string-pair to be passed to solver.

Description Syntax: 'PARAM-STRING "some-name" VALUE "some-value"' (Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system.)

20.4.20 Polynomial Order

Scope: Aztec Equation Solver

`Polynomial Order {= | are | is}` *`polynomial_order`*

Parameter	Value	Default
polynomial_order	integer	undefined

Summary Polynomial order of preconditioning method.

20.4.21 Preconditioner Subdomain Overlap

Scope: Aztec Equation Solver

`Preconditioner Subdomain Overlap {= | are | is}` *`Overlap`*

Parameter	Value	Default
Overlap	integer	undefined

Summary Ovrlap of Schwarz subdomains (eg, 0,1 or 2).

20.4.22 Preconditioning Method

Scope: Aztec Equation Solver

`Preconditioning Method {= | are | is}` *`AztecPrecondMethods`*

Parameter	Value	Default
AztecPrecondMethods	{dd-bilu | dd-icc | dd-ilu | dd-ilut | dd-lu | dd-rilu | jacobi | least-squares | neumann | none | symmetric-gauss-seidel}	undefined

---

Summary Selection of the equation preconditioning method.

Description Valid values for this are contained in the AztecPrecondMethods enum.

20.4.23 Preconditioning Steps

Scope: Aztec Equation Solver

`Preconditioning Steps {= | are | is}` *`preconditioning_steps`*

Parameter	Value	Default
preconditioning_steps	integer	undefined

Summary Number of Jacobi, Gauss-Seidel, or other preconditioning methods' applications per iteration.

20.4.24 Residual Norm Scaling

Scope: Aztec Equation Solver

`Residual Norm Scaling {=|are|is}` *`AztecResidualNormScaling`*

Parameter	Value	Default
AztecResidualNormScaling	{anorm|none|r0|rhs}	undefined

Summary Scaling method for the residual norm.

Description Determines the residual expression used in convergence checks and printing.

20.4.25 Residual Norm Tolerance

Scope: Aztec Equation Solver

`Residual Norm Tolerance {=|are|is}` *`residual_norm_tolerance`*

Parameter	Value	Default
residual_norm_tolerance	real	undefined

Summary Iterative solution method relative residual convergence tolerance.

20.4.26 Restart Iterations

Scope: Aztec Equation Solver

`Restart Iterations {=|are|is}` *`restart_iterations`*

Parameter	Value	Default
restart_iterations	integer	undefined

Summary Number of iterations between GMRES restarts.

20.4.27 Shared Ownership Rule

Scope: Aztec Equation Solver

`Shared Ownership Rule {=|are|is}` *`SharedOwnershipRule`*

Parameter	Value	Default
SharedOwnershipRule	{low-numbered-proc|proc-with-local-elem|sierra_specifies}	undefined

---

Summary Controls the way owning processors are chosen for shared nodes in the FEI.

Description 'low-numbered-proc' is the default.

'proc-with-local-elem' is another valid value.

'sierra_specifies' is the other valid value

20.4.28 Solution Method

Scope: Aztec Equation Solver

`Solution Method {=|are|is}` *`AztecSolverMethods`*

Parameter	Value	Default
AztecSolverMethods	{bicgstab | cg | cgs | gmres | lu | tfqmr}	undefined

Summary — Selection of the linear-system solution algorithm.

Description — The AztecSolverMethods enum contains valid values for this line-command. They are mostly iterative methods (Krylov subspace algorithms), except for 'lu' which is a serial direct solver provided by the 'Y12M' library.

20.4.29 Select Fei

Scope: Aztec Equation Solver

`Select Fei {=|are|is}` *`WhichFEI`*

Parameter	Value	Default
WhichFEI	{new | old}	undefined

Summary — Selection of old vs new fei.

Description — This parameter will be deprecated, it is used as a transition aid during some FEI refactoring.

20.5 Gdsw Equation Solver

Scope: Sierra

```
Begin Gdsw Equation Solver Solver Name
    Debug Output Level {=|are|is} Level
    Debug Output Path {=|are|is} DebugOutput
    Fei Error Behavior {=|are|is} FeiErrorBehavior
    Fei Output Level {=|are|is} FeiOutputLevels
    Maximum Iterations {=|are|is} max_iterations
    Param-Int Parameter_name Value integer_value
    Param-Real Parameter_name Value Real_value
    Param-String Parameter_name Value String_value
    Residual Norm Tolerance {=|are|is} residual_norm_tolerance
End
```

Summary — A set of solver parameters for GDSW equation solver.

20.5.1 Debug Output Level

Scope: Gdsw Equation Solver

`Debug Output Level {= | are | is}` *Level*

Parameter	Value	Default
Level	`integer`	`undefined`

Summary Output level for debugging. Generally 0 means no solver screen output, and higher values of this parameter correspond to more screen output.

20.5.2 Debug Output Path

Scope: Gdsw Equation Solver

`Debug Output Path {= | are | is}` *DebugOutput*

Parameter	Value	Default
DebugOutput	`string`	`undefined`

Summary Specify path where debug-logs and other debug output files will be placed.

20.5.3 Fei Error Behavior

Scope: Gdsw Equation Solver

`Fei Error Behavior {= | are | is}` *FeiErrorBehavior*

Parameter	Value	Default
FeiErrorBehavior	`{abort	returncode}`	`undefined`

Summary Set FEI error behavior to abort (rather than the default which is to simply print a message and return an integer error code).

Description This is becoming less relevant starting with FEI version 2.11, as the FEI begins to adopt exception handling and abandon the antiquated int-error-code interfaces.

20.5.4 Fei Output Level

Scope: Gdsw Equation Solver

`Fei Output Level {= | are | is}` *FeiOutputLevels*

Parameter	Value	Default
FeiOutputLevels	`{all	brief_logs	full_logs	matrix_files	none	stats}`	`undefined`

Summary Control the amount of output produced by FEI.

Description Output level controls the amount of debugging information (screen output, matrix/vector files and log files) produced by FEI. The FeiOutputLevels enum contains the valid values for this line-command. The location of files can be controlled by the 'debug output path' line-command.

20.5.5 Maximum Iterations

Scope: Gdsw Equation Solver

`Maximum Iterations {=|are|is}` *`max_iterations`*

Parameter	Value	Default
max_iterations	integer	undefined

Summary Maximum number of solution method iterations.

20.5.6 Param-Int

Scope: Gdsw Equation Solver

`Param-Int` *`Parameter_name`* `Value` *`integer_value`*

Parameter	Value	Default
Parameter_name	"string"	undefined
integer_value	integer	undefined

Summary String-Key/Integer-Value pair to be passed to solver.

Description Syntax: 'PARAM-INT "some-name" VALUE 123' Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system. Can also be used to pass parameters to Aztec, e.g.:

PARAM-REAL "AZ_ilut_fill" VALUE 3

20.5.7 Param-Real

Scope: Gdsw Equation Solver

`Param-Real` *`Parameter_name`* `Value` *`Real_value`*

Parameter	Value	Default
Parameter_name	"string"	undefined
Real_value	real	undefined

Summary String-Key/Real-Value pair to be passed to solver.

Description Syntax: 'PARAM-REAL "some-name" VALUE 1.23' Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system. Can also be used to pass parameters to Aztec, e.g.:

PARAM-REAL "AZ_drop" VALUE 1.e-8

20.5.8 Param-String

Scope: Gdsw Equation Solver

`Param-String` *`Parameter_name`* `Value` *`String_value`*

Parameter	Value	Default
Parameter_name	`"string"`	`undefined`
String_value	`"string"`	`undefined`

Summary — Key/Value string-pair to be passed to solver.

Description — Syntax: 'PARAM-STRING "some-name" VALUE "some-value"' (Primarily used for passing values to the ML package. ML has a large number of parameters which haven't been individually incorporated into our parameter-parsing system.)

20.5.9 Residual Norm Tolerance

Scope: Gdsw Equation Solver

`Residual Norm Tolerance {=|are|is}` *`residual_norm_tolerance`*

Parameter	Value	Default
residual_norm_tolerance	`real`	`undefined`

Summary — Iterative solution method relative residual convergence tolerance.

Chapter 21

Postprocessing Operations

21.1 Overview

Aria supports postprocessing of solution information partially through Aria itself and alternatively through the Encore library [34]. Native postprocessing through Aria is invoked using a single command line, for example:

```
Postprocess density on block_1
```

This creates a Field variable "pp->density" that is available for output from the Results Output Command Block 25.1. Any expression created during the calculation may be "**POSTPROCESS**"ed. Of particular interest are properties defined in the material block input 4 and source terms present for any equation. In addition single components of any tensor quantity can be postprocessed by appending the suffix for the desired component (e.g. "_XX", "_XY", etc.) to the end of the expression name used in the "Postprocess" line command. Similarly, the Frobenius norm of a tensor may be postprocessed by using the "_NORM" suffix.

In addition to the *pp* postprocessing command another means of postprocessing called *Solution Options* is also available. Whereas *pp* can be used for material parameters, the use of *Solution Options* usually implies that the postprocessing variable is the result of an algorithmic computation. Use of postprocessing will follow two distinct patterns, depending upon whether or not Equation Systems 17.1 are employed in the simulation.

When Equation Systems are not being used then within the input file postprocessing command lines will appear at the Region scope as illustrated below.

```
 .
 Begin Procedure My_Aria_Procedure
  .
  Begin Aria Region My_Region
   .
   postprocess ENERGY_SOURCE on block_2
   .
   Begin Solution Options
    .
    post process nodal normalized HEAT_TRANSFER_COEFFICIENT on surface_5 as h5
    .
   End Solution Options
   .
  End Aria Region My_Region
     .
 End Procedure My_Aria_Procedure
```

.

When Equation Systems are being used in a simulation then postprocessing command lines must appear within scope of the Equation System in which the quantities to be postprocessed are available/computed. As an example, a convective flux boundary condition may lie within an Equation System Energy hence postprocessing of the heat transfer coefficient would be defined within that scope.

When using postprocessing commands specific to *solution options* the **SOLUTION OPTIONS** command block must appear at the region scope. Once the Solution Options command block is present then individual solution options commands can appear within the appropriate Equation System. As an example suppose one wishes to include a general specification of quantities generally defined for the entire simulation, i.e. mass and volume, as well as a normalized heat transfer coefficient. Here mass and volume are processed at processed at the Region scope and the heat transfer postprocessing is done at Equation System scope as illustrated below. Note that the

```
.
Begin Procedure My_Aria_Procedure
 .
 Begin Aria Region My_Region
  .
  Begin Equation System Energy
   .
   postprocess ENERGY_SOURCE on block_2
   .
   # no Begin Solution Options required
   post process nodal normalized HEAT_TRANSFER_COEFFICIENT on surface_5 as h5
   .
  End Equation System Energy
   .
 End Aria Region My_Region
  .
 Begin Solution Options
  .
  post process volume
  post process mass
  .
 End Solution Options
 .
End Procedure My_Aria_Procedure
 .
```

Note that for simulations with Equation Systems where physics related postprocessing is desired but no general postprocessing is required, i.e. no mass or volume calculation, the Begin Solution Options command block must be appear but but it can be empty.

Aria postprocessing with Encore is usually accomplished using Encore specific commands [34]. Input commands for both *pp* and *solution options* postprocessing are included in what follows.

21.1.1 Postprocess

Scope:

Postprocess *Expression* [{of | species} *SpeciesName* | {in | material_phase} *MaterialPhaseName*]{@ | at | for | in | on | over} *MeshPartIdentifier* [Using *ElementType* | Averaging Method {= | are | is} *AveragingMethod*]

Parameter	Value	Default
Expression	string	undefined
SpeciesName	string	undefined
MaterialPhaseName	string	undefined
MeshPartIdentifier	string	undefined
ElementType	string	undefined

Summary

Postprocess Expression on a mesh part (block, surface, etc.). By default, the quantity is computed on the grid element nodes. If the optional arguments "using ElementType" are provided, the postprocessed field will be computed on that element. If the ElementType is P0 then the quantity is computed as a constant element value. By default the nodal field value is calculated using a weighted average of the value in each element at the node location where the element contributions are weighted by $\int w_{node} d\Omega$, This is the SUPPORT_WEIGHTED averaging method. Two alternative averaging methods are available. SUPPORT_AVERAGED integrates $\int \phi w_{node} d\Omega$ (where ϕ is the expression being postprocessed) over all supporting elements and normalizes by $\int w_{node} d\Omega$ over all supporting elements. SIMPLE just takes the mean of the element values at the node location.

21.2 Solution Options

Scope: Aria Region

```
Begin Solution Options OptionsName

    Check Matrix For Discrete Maximum Principle

    Post Process PostProcessType VariableName On IoPartList...  As OutputName [ Using
    WeightList...  Weight ]

    Post Process Flux fluxName On ioPartList...  As outputName [ Restricted To restrictPartList...
    ]

    Post Process Mass

    Post Process Nodal PostProcessType VariableName On IoPartList...  As OutputName [
    Using WeightList...  Weight ]

    Post Process Volume

    Begin Cvfem Algorithm Specification PStabName
    End

    Begin Edc Model Specification EdcSpecName
    End

    Begin Hdiff Model Specification HdiffOptionsName
    End

    Begin Porous Flow Options blockName
    End
```

```
    Begin Species Options blockName
    End

    Begin Turbulence Model Specification TurbSpecName
    End

End
```

Summary Specify information regarding the governing equations to be solved.

21.2.1 Check Matrix For Discrete Maximum Principle

Scope: Solution Options

Summary Examine rows of the system matrix before boundary conditions are applied and check if the diagonal entries are non-negative, the off-diagonal entries are non-positive, the row-sum is non-negative. If these conditions fail, it is possible the solution will contain non-physical values, and the discrete maximum principle is not satisfied by the matrix. If any of these conditions fail, a warning is printed to the log file. More output can be obtained with turning on debug logging.

21.2.2 Post Process

Scope: Solution Options

Post Process *PostProcessType VariableName* On *IoPartList...* As *OutputName* [Using *WeightList...* Weight]

Parameter	Value	Default
PostProcessType	{integral | normalized}	undefined
VariableName	string	undefined
IoPartList	string...	undefined
OutputName	string	undefined

Summary Post process an Aria variable and save it to a global variable.

Description Post process Aria variable VariableName and save it to a global variable OutputName. The global OutputName will be written to the log file and is also available for output to an ExodusII database. During post processing the variable a product is formed from VariableName and the evaluation of WeightList items. The keyword WEIGHT terminates the Weightlist items.

21.2.3 Post Process Flux

Scope: Solution Options

Post Process Flux *fluxName* On *ioPartList...* As *outputName* [Restricted To *restrictPartList...*]

Parameter	Value	Default
fluxName	`string`	`undefined`
ioPartList	`string...`	`undefined`
outputName	`string`	`undefined`

Summary Post process an Aria flux and save the integrated results to a global Aria Field.

Description Post process Aria flux FluxName and save the integrated flux to a global Field variable OutputName. The resultant variable OutputName will be available for output to an ExodusII database. Examples of FluxName are heat_conduction and current_density.

21.2.4 Post Process Mass

Scope: Solution Options

Summary Post process both the total mass for all blocks and per-block mass. By default these values will be output to log file.

Additionally these values are available as global variables which can be output to the ExodusII database. Naming convention of these values are TOTAL_MASS for the aggregate mass and MASS_mesh_entity_name (e.g. MASS_BLOCK_1) for individual blocks.

21.2.5 Post Process Nodal

Scope: Solution Options

`Post Process Nodal` *`PostProcessType VariableName`* `On` *`IoPartList...`* `As` *`OutputName`* `[ Using` *`WeightList...`* `Weight ]`

Parameter	Value	Default
PostProcessType	`{integral | normalized}`	`undefined`
VariableName	`string`	`undefined`
IoPartList	`string...`	`undefined`
OutputName	`string`	`undefined`

Summary Post process an Aria nodal variable and save the results to another Aria Field.

Description Post process Aria variable VariableName and save it to another nodal Field variable OutputName. The resultant variable OutputName will be available for output to an ExodusII database. PostProcessType is NORMALIZED so that during post processing the variable a product is formed from VariableName and the evaluation of WeightList items. The keyword WEIGHT terminates the Weightlist items.

21.2.6 Post Process Volume

Scope: Solution Options

Summary Post process both the total volume and and per-block volume. By default these values will be output to log file.

Additionally these values are available as global variables which can be output to the ExodusII database. Naming convention of these values are TOTAL_VOLUME for the aggregate volume and VOLUME_mesh_entity_name (e.g. VOLUME_BLOCK_1) for individual blocks.

21.3 Residual Based Integrated Surface Flux Calculations

Oftentimes one needs to require the evaluation of surface flux over some portion of the problem domain. Examples of such evaluations include the net thermal energy through a surface or the net force on a body. While the evaluation can be carried out in a variety of ways one method of carrying out an accurate evaluation of these quantities is to employ a residual approach. The general approach is described here for problems involving scalar variables but the technique applies equally as well to vector variables.

Governing partial differential equations for scalar potential u are often of the form

$$C\frac{\mathtt{D}u}{\mathtt{D}t} = \nabla \cdot \mathbf{q} + Q \tag{21.1}$$

where C is storage coefficient, $\mathbf{q}$ is the flux of u, Q is a volumetric source and the substantial derivative $\mathtt{D}()/\mathtt{D}t$ accounts for material motion. The flux of u is oftentimes a simple function of the potential gradient

$$\mathbf{q} = -k \nabla u \tag{21.2}$$

where k is a diffusion tensor. In many cases k is isotropic so it is treated as a constant coefficient.

To obtain solutions of scalar PDE (21.1) using finite elements, one forms a weighted residual over a spatial discretization of elements of the problem domain Γ. Integration by parts, then yields a general form of the nodal residual

$$R^i = \int_\Omega \left[\left(C\frac{Du}{Dt} - Q \right) \phi^i - \mathbf{q} \cdot \nabla \phi^i \right] d\Omega + \int_\Gamma \mathbf{n} \cdot \mathbf{q}\, \phi^i d\Gamma \tag{21.3}$$

where i denotes a node of the meshed discretization, ϕ^i is the finite element weight function for node i. The collection of nodal residual equations over Ω form a linear system of equations for which we seek to minimize the nodal residuals, $R^i = 0$ over the solution domain.

Dirichlet boundary conditions on (21.3) for specific nodes can be eliminated from the system and the unknown temperature is assigned directly. For flux boundary conditions, the term $\mathbf{n} \cdot \mathbf{q}$ is replaced with a model of the surface normal flux. Surface contact boundary conditions can also be accounted for through a special implementation of this surface integral as outlined in the Aria user manual.

In some problems there is need of computing the surface flux f_n over a subset of the domain boundary Γ and one might consider evaluating

$$f_n = \int_{\Gamma_n} \mathbf{n} \cdot \mathbf{q}\, d\Gamma_n \quad . \tag{21.4}$$

This approach requires the evaluation of finite element gradients on element boundaries as per (21.2) but is known to be problematic since gradients of the finite element solutions in terms of the basis functions ϕ^i are of lower order than the finite element solution. An alternative is to evaluate the flux over Γ_n using the volumetric term of (21.3)

$$f_n^i = -\int_{\Omega_n} \left[\left(C\frac{Du}{Dt} - Q \right) \phi^i - \mathbf{q} \cdot \nabla \phi^i \right] d\Omega_n \tag{21.5}$$

which must be computed while forming the linear system of equations and is known to be second order accurate. Note that f^i is the integrated surface flux contribution at a node, not a flux per unit area. The area over which f_n^i is computed is of course mesh dependent and is given by the finite element support for node i.

In order to obtain the nodal flux (21.5), we proceed with our typical FEM assembly procedure in the evaluation of (21.3) where the volume and surface contributions are handled in separate portions of the code. Using the `Save Residuals = Before_BCs` input command line in Aria, we request that intermediate values of the residuals be stored after the volumetric terms are calculated but before the surface contributions are computed. These integrated nodal flux values are stored in a field named `residual->dof_name`, where `dof_name` corresponds to the solution variable u for a given equation.

The total flux on a portion of the boundary Γ_n Computation of t is illustrated for the energy equation with solution variable `temperature`. is obtained by summing the residual values for all nodes along the surface

$$f_n = -\sum_{i}^{\in\Gamma_n} - \int_{\Omega_n} \left[\left(C\frac{Du}{Dt} - Q \right) \phi^i - \mathbf{q} \cdot \nabla\phi^i \right] d\Omega_n \ . \tag{21.6}$$

Computation of the total flux over Γ_n is illustrated for the energy equation with solution variable `temperature`. To perform this calculation using Aria one first needs to enable the saving of residuals. In the Aria Region block we add the command line

```
Save Residuals = Before_BCs
```

This instructs Aria to store the result of (21.3) in a field named `residual->temperature`. Next, to perform the summation in (21.6) on a portion of the surface, say `sideset_12` one can add an Encore summation postprocessor block at the domain scope:

```
Begin Summation Postprocessor flux_12
 Use Function residual->temperature
 Surfaces surface_12
End
```

To execute this postprocessor, one adds an additional command line to the Aria Region scope:

```
Evaluate Postprocessor flux_12
```

This will create a global variable named `flux_12` whose result will be written to the Aria log file for each successful solution step. Finally, one can output the result `flux_12` to the Exodus results output database file by adding the following line to the appropriate output block:

```
global variables = flux_12
```

Similarly the global variable can be written to the heartbeat output file by adding the line

```
variable = global flux_12
```

to the Heartbeat file command block.

Chapter 22

Enclosure Radiation Reference

22.1 Enclosure Radiation

When energy radiates from one portion of a surface to another, and the intermediate medium is transparent(i.e., it does not absorb any energy), then enclosure radiation may be used to model the heat flux on the surface. Using the net radiation method [35], the normal flux at a particular location on the surface may be written as the difference between the emitted radiative heat flux leaving the surface, and the absorbed incident radiative flux due to the rest of the enclosure, namely

$$q_n = \sigma \epsilon T^4 - \alpha G, \tag{22.1}$$

where T is the temperature, α denotes the absorptivity of the surface, and G represents the surface irradiation. Under the additional assumption that the emissivity, absorptivity, and reflectivity are independent of direction and wavelength, we may write

$$\epsilon = \alpha = 1 - \rho, \tag{22.2}$$

where we have used the conventional symbol ρ for reflectivity. In this section, ρ always refers to reflectivity and not density.

Without loss of generality, we can regard the enclosure, $\Gamma_{\mathcal{E}}$, as a union of E surfaces,

$$\Gamma_{\mathcal{E}} = \Gamma_1 \cup \Gamma_2, \ldots \Gamma_{E-1} \cup \Gamma_E$$

This situation is illustrated in Figure 22.1, where the the radiosity for surface i in the enclosure is defined to be

$$J_i = \sigma \epsilon_i T_i^4 + \rho_i G_i, \tag{22.3}$$

where u_i is the spatially constant temperature on Γ_i. The surface irradiation for surface i is determined by the radiosity of all the other surfaces in the enclosure through the relation

$$G_i = \sum_{j=1}^{E} F_{ij} J_j, \tag{22.4}$$

where F_{ij} denotes the geometric viewfactor of surface i with respect to surface j. The viewfactor may be considered the fraction of energy that leaves surface i and arrives at surface j.

Upon substitution of equations (22.4) and (22.2) into (22.3), the radiosity may be written as

$$J_i - (1 - \epsilon_i) \sum_{j=1}^{N} F_{ij} J_j = \sigma \epsilon_i T_i^4, \tag{22.5}$$

Finally, the first J_i term in (22.5) may be moved inside the summation to yield

$$\sum_{j=1}^{N} \left[\delta_{ij} - (1 - \epsilon_i) F_{ij} \right] J_j = \sigma \epsilon_i T_i^4, \tag{22.6}$$

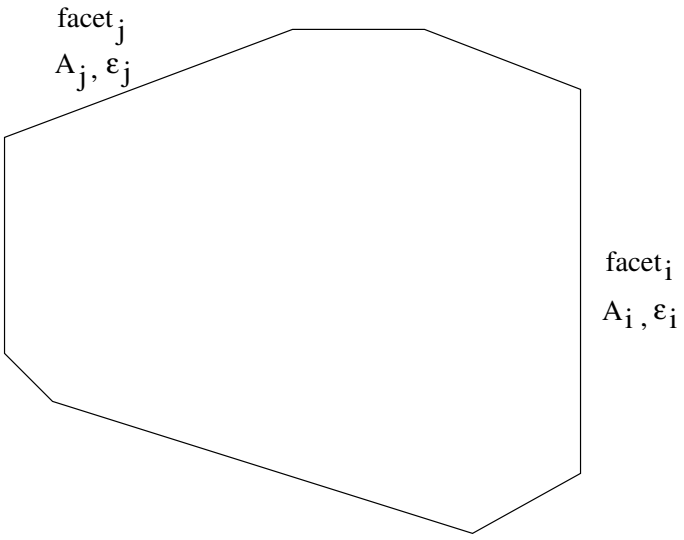


Figure 22.1. Two arbitrary facets radiating energy to one another in a *radiation enclosure*. The energy exchanged depends on: the shape, orientation, distance, area A_i, A_j, temperatures T_i, T_j, and radiative properties of the facets ϵ_i, ϵ_j.

where

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{22.7}$$

(22.6) is a nonlinear system of equations for the radiosities that must be solved simultaneously with (at least) the energy transport equations (see 5.3). Finally, we may rewrite (22.1) to express the normal flux boundary condition on surface i as

$$q_n = \sigma \epsilon T^4 - \epsilon G_i, \tag{22.8}$$

where G_i is given by (22.4).

22.2 Enclosure Radiation Surface Flux

The enclosure radiation boundary condition given by (22.1) enters our weak statement in a manner similar to that of the surface radiation flux described in 9.2.19 The enclosure radiation flux was written as (22.1), which we repeat here for convenience:

$$q_n(T) = \sigma \epsilon T^4 - \epsilon G, \tag{22.9}$$

This flux and the surface radiation flux are both nonlinear, and vary as the fourth power of the temperature, but there are important differences. First, instead of a known reference temperature T_r, the surface irradiation, G, appears in (22.9), and must be calculated as part of the solution process. To begin, let $\Gamma^e_{\mathcal{E}}$ be an arbitrary finite element on the enclosure. We call $\Gamma^e_{\mathcal{E}}$ a *facet*, and let $\Gamma_{\mathcal{E}}$ consist of a total of E facets. Furthermore, the flux (22.9) is constant on a facet; to emphasize this, let us rewrite it as

$$q^e_n(T) = \sigma \epsilon \left[T^e\right]^4 - \epsilon G_e(T^e), \tag{22.10}$$

where T^e is some average facet temperature, and $G_e(T^e)$ is the irradiation on facet e, which is determined by the combined effects of all the other facets in the enclosure. Recall that this was expressed in Section 22.1

as

$$G_e(T^e) = \sum_{f=1}^{E} F_{ef} J_f(T^e). \tag{22.11}$$

Details of our nonlinear solution strategy are not included here but it is important to note here that the unknown radiosities, J_f, depend upon the solution of the temperature. We use a decoupled approach, and calculate the viewfactors, F_{ef}, and radiosities, J_f, via the Chaparral library [36]. Recall that the radiosities are obtained by solving the system of equations

$$\sum_{f=1}^{N} \left[\delta_{ef} - (1-\epsilon_e)F_{ef}\right] J_f = \sigma\epsilon_e \left[T^e\right]^4 \tag{22.12}$$

In (22.12), it is important to realize that the J_f and T^e are constant values associated with a given facet. But in our finite element approximation, the temperatures are associated with the nodes. Therefore, the facet temperature should be considered a projection, or averaging of the temperature found at the nodes of a given facet. Hence, we define the facet temperature as

$$T^e = \frac{\sum_{i=1}^{N} T_i \int_{\Gamma_{\mathcal{E}}^e} \psi_i^e \, d\Gamma}{\int_{\Gamma_{\mathcal{E}}^e} d\Gamma} \tag{22.13}$$

It is now convenient to define the projection vector $\mathbf{P}^e$, where row i is defined by

$$P_i^e = \frac{1}{A^e} \int_{\Gamma_{\mathcal{E}}^e} \psi_i^e \, d\Gamma, \tag{22.14}$$

and A^e is the area of facet e. Thus, (22.13) may be written as

$$T^e = \sum_{i=1}^{N} P_i^e T_i \tag{22.15}$$

22.3 Solution Strategy

When using Newton's method one can employ the surface radiation flux 22.9 directly to obtain Jacobian and residual contributions to the linear system of equations. Solution convergence of enclosure radiation problems by successive substitution is improved by employing a linearization of the surface radiation flux as described in what follows.

Now that we have appropriately defined the facet temperature, we may linearize the flux given in (22.10). Expanding $q_n(T)$ in a Taylor series about a known state T_*^e, we obtain

$$q_n\left(T_* + \delta T\right) = \sigma\epsilon \left[T_*^e\right]^4 - \epsilon G_e(T_*^e) + 4\sigma\epsilon \left[T_*^e\right]^3 \delta T \tag{22.16}$$

where $\delta T = T^e - T_*^e$, and we have chosen to evaluate the irradiation at the known state, T_*^e. If we collect terms involving the known T_*^e and unknown T^e, (22.16) may be rearranged and written as

$$q_n\left(T_* + \delta T\right) = 4\sigma\epsilon \left[T_*^e\right]^3 T^e - 3\sigma\epsilon \left[T_*^e\right]^4 - \epsilon G_e(T_*^e) \tag{22.17}$$

In order to apply this flux in our weak statement, we must integrate it against the test function, ψ_i^e. Therefore, we have

$$\int_{\Gamma_{\mathcal{E}}^e} \psi_i^e q_n(T) \, d\Gamma = \int_{\Gamma_{\mathcal{E}}^e} \psi_i^e \left\{ 4\sigma\epsilon \left[T_*^e\right]^3 T^e - 3\sigma\epsilon \left[T_*^e\right]^4 - \epsilon G_e(T_*^e) \right\} d\Gamma \tag{22.18}$$

Exploiting the fact that all quantities on a given facet are constant, except for the shape functions, (22.18) becomes

$$\begin{aligned}\int_{\Gamma_{\mathcal{E}}^e} \psi_i^e q_n(T)\, d\Gamma = &\int_{\Gamma_{\mathcal{E}}^e} \psi_i^e\, d\Gamma \left\{4\sigma\epsilon \left[T_*^e\right]^3 T^e\right\} \\ &- \int_{\Gamma_{\mathcal{E}}^e} \psi_i^e\, d\Gamma \left\{3\sigma\epsilon \left[T_*^e\right]^4 + \epsilon G_e(T_*^e)\right\}\end{aligned} \tag{22.19}$$

If we introduce the definitions (22.15) and (22.14) into (22.19), we see that the linearized form of the element matrix contribution is

$$K_{ij}^e = 4\sigma\epsilon \left[T_*^e\right]^3 A^e P_i^e P_j^e \tag{22.20}$$

and the contribution to the source vector is

$$F_i^e = A^e \epsilon \left\{3\sigma \left[T_*^e\right]^4 + G_e(T_*^e)\right\} P_i^e \tag{22.21}$$

We remark that, when calculating terms in Equations (22.20) and (22.21), experience has shown that it is important to project the exponential power of the temperature, as opposed to projecting the temperature, then raising it to some exponential power. In other words, spurious oscillations are less likely to occur if

$$[T^e]^4 = \sum_{i=1}^{N} P_i^e T_i^4$$

is used instead of

$$[T^e]^4 = \left[\sum_{i=1}^{N} P_i^e T_i\right]^4$$

22.4 Banded Wavelength Enclosure Radiation

The enclosure radiation model previously presented considers a net response over all wavelengths assuming that all surfaces behave as grey bodies. Here the surface response with regard to different wavelengths is implicitly included in the model by using temperature dependent emissivities and allowing the surfaces to emit as grey bodies. In this case the surface flux previously mentioned 22.1 can be alternatively expressed as

$$q_n = \int_0^\infty q(\lambda)\ \ d\lambda\ . \tag{22.22}$$

In many applications engineering materials respond differently to different portions of the thermal energy spectrum. Thus modeling the thermal radiation response of these materials using the entire blackbody spectrum results in poor characterization of the enclosure response. For this purpose more specialized approaches for radiation modeling of wavelength dependent surfaces have been developed using continuous representations of emissivity as a function of wavelength and temperature while integrating over the wavelength spectrum. For numerical modeling the simplest approach involves discretization of the wavelength spectrum into a few representative bands N_b and integrating over each of the k bands. Using this approach the methods previously described follow directly except that the emissivity and emissive power now have independent representations in a wavelength band, each of which contributes some surface flux Δq_k thus

$$q_n = \sum_{k=1}^{N_b} \Delta q_{k\lambda}\ . \tag{22.23}$$

Recalling that flux can also be expressed in terms of radiosity then similarly the radiosities are obtained by solving a system of equations

$$\sum_{k=1}^{N_b}\sum_{f=1}^{N}\left[\delta_{(ek)f}-(1-\epsilon_{(ek)})F_{ef}\right]\Delta J_{fk}\Delta\lambda_k=\sum_{k=1}^{N_b}\mathrm{F}_{\Delta\lambda_k T}\sigma\epsilon_{(ek)}\left[T^e\right]^4\Delta\lambda_k \tag{22.24}$$

where ΔJ_{fk} is the incremental radiosity and $\mathrm{F}_{\Delta\lambda_k}$ is the blackbody fraction for band k. Finally we note that given a fixed facet temperature field this solution can be carried out independently for each wavelength band and the radiosities ΔJ_{fk} can be accumulated to obtain the net facet radiosity J_f and net flux q_n. Thus the banded wavelength model very much resembles the more simplified enclosure radiation model 22.1. From a simulation point of view, the major difference being that the modeler must supply information describing the band discretization in addition to emissivity models for each band.

Construction of the banded wavelength model begins by first prescribing the emissivity variation for each surface of the enclosure as a function of wavelength λ. The overall band discretization is then obtained by collective consideration of the wavelengths at which the emissivity changes on any of the given surfaces. Thus the number of bands for the enclosure will likely be more than those for any single surface and is demonstrated in Figure 22.2 below. It is important to note that for each modeled band an emissivity model must be supplied for each surface, even when its emissivity is not changing.

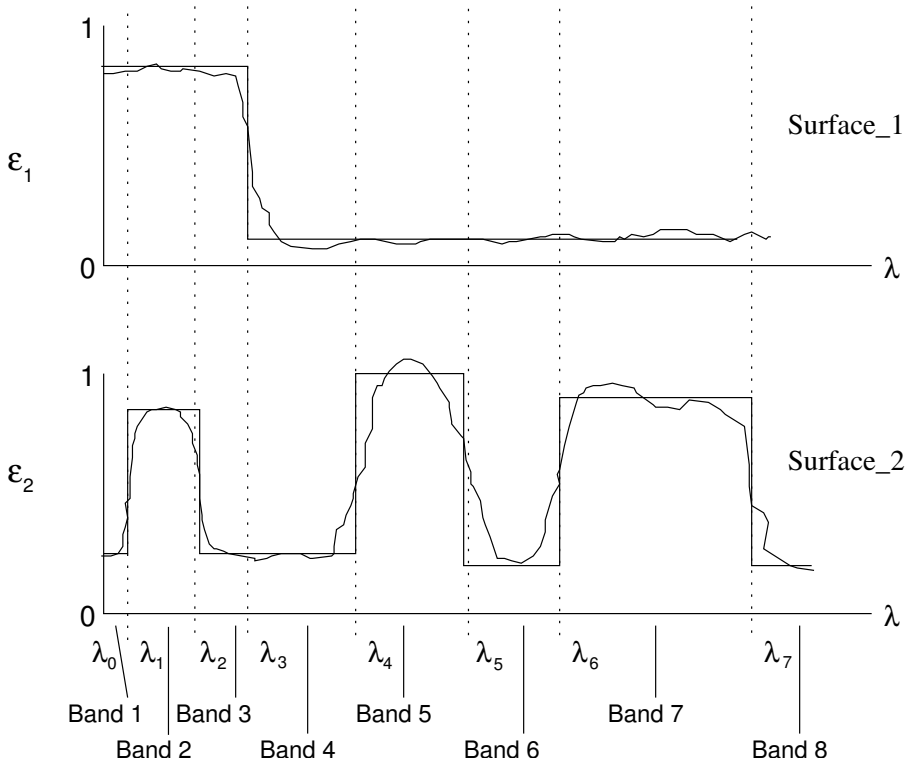


Figure 22.2. Emissivity Bands For Two-Surface Problem

22.5 Enclosure Radiation Modeling

Enclosure radiation modeling is normally performed in several steps,

- Compute the viewfactors or read in pre-computed viewfactors.
- Radiosity solve.
- Perform the coupled finite element solve.

The last step is handled implicitly once boundary conditions on the finite element model have been defined. To facilitate the first three steps the analyst must first define the enclosure. For complex enclosures smoothing (conditioning) of the viewfactors is often required to ensure the quality of the viewfactors before proceeding to the radiosity solve and a facility for performing this operation is also provided. As part of this enclosure definition details concerning both computation of viewfactors, and the radiosity solve must be included. Full details of enclosure definition, viewfactor calculation, viewfactor smoothing and radiosity solve are specified in separate input command blocks. Additional input command blocks must also be supplied in order to employ the banded-wavelength enclosure model. Individual directives appearing within the aforementioned command blocks are described in the sections which follow.

In all cases the enclosure is viewed as a closed surface. As illustrated in Figure 22.3 the discretization may need to be modified in order to properly define the closed surface.

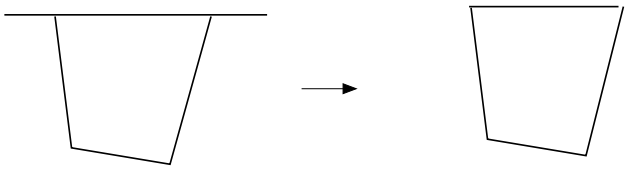


Figure 22.3. Discretization and Associated Closed Enclosure Surface.

For problems in which the enclosure surface is not entirely meshed, it can be implicitly closed by defining a partial enclosure and its associated properties within the enclosure definition command block. The partial enclosure is demonstrated in Figure 22.4. Note that the partial enclosure approach is a model simplification and is no substitute for a fully-meshed enclosure. Best results with partial enclosures are obtained when most of the enclosure model facets have the same view of the partial enclosure area and temperature.

As part of the partial enclosure description one must supply a partial enclosure area but for many problems it may be difficult to initially compute or even estimate the partial enclosure area. For these cases provision is made within Aria to internally compute the minimum area that should be used. To obtain the minimum area one temporarily assigns a small number for the partial enclosure, starts the problem and terminates it after one step. The minimum partial enclosure area can then be obtained from the .output file. Further discussion of the partial enclosure is given in a section which follows 22.5.1.

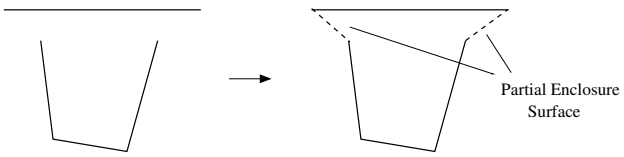


Figure 22.4. Discretization and Associated Partial Enclosure Surface.

It is worth mentioning that the viewfactor between two facets is roughly proportional to $1/r^2$ where r is the distance between facet centroids. Thus it follows directly that facets which are nearly parallel may adversely affect the viewfactor calculation. Similar statements apply to poorly equivalenced surface intersections leading to slivered surfaces becoming part of an enclosure. In the worst of circumstances the

viewfactor will simply fail when the distance between facets is too small and the mesh may need to be repaired.

Beta Capability: The DASH Enclosures is not considered a production capability as the setup differs for different processor counts.

At the user's request some attempt can be made to remedy these situations by preprocessing of the surface facets to remove these defects, via an algorithm denoted as DASH. This algorithm can be applied universally to all the enclosures or to each enclosure independently. The use of DASH for the construction of enclosures is functionally equivalent to the standard prescription of surfaces except in the case where partial enclosures are present. DASH presents a powerful mechanism for automatically generating enclosures but the user has to ensure that the problem being addressed is appropriate such as cases where the entire set of element blocks is used to generate the enclosures but the user is only interested in a subset.

One measure of how accurately the viewfactors are being computed is the rowsum value. For each surface facet in a fully-closed enclosure The rowsum value should approach one, thus the total rowsum value is always the number of facets. In cases where the target rowsum is not equal to one the surface may not be fully-closed or the view factor calculation may be inaccurate for some other reason. In either event the analyst should consider investigate possible causes of the discrepancy. The viewfactor calculation is often followed by a smoothing process which may offset some of the errors indicated by the rowsum values. Note that while use of the Dash algorithm can be useful to overcome inaccuracies, there is no substitute for repair of any mesh defects.

Spatial discretization of the enclosure consists of surfaces as illustrated in Figure 22.1 where the surfaces respect an inward facing normal orientation. For most cases the enclosure is devoid of interior mesh discretization. In the event that the enclosure interior is meshed, the interior mesh can be ignored by using the OMIT_VOLUME command line in the FINITE_ELEMENT_MODEL command block.

Beta Capability: Overlapping enclosures are not fully tested and are not considered to be a production capability

The objective in solving the enclosure radiation problem is to obtain flux values at the enclosure cavity surfaces for later application to the finite element model. In a special class of closed enclosure problems one may wish to superpose flux contributions from different enclosure radiation problems overlying the same surfaces as depicted in Figure 22.5. This situation often arises in cases where the enclosure thermal response is wavelength dependent. For any given enclosure, unique descriptions of the radiation enclosure surfaces are provided to the Chaparral library. When enclosure radiation surfaces overlap special provision must be made in order to uniquely define these surfaces for the Chaparral library. Here the distinction between surfaces is made by denoting an enclosure overlying another enclosure as an OVERLAPPING ENCLOSURE. In the example, two consistent definitions of OVERLAPPING ENCLOSURE are possible. Enclosure 2 can be denoted as the overlapping enclosure since it overlaps enclosures 1 and 3. Likewise enclosures 1 and 3 could be denoted as overlapping enclosures since they overlap Enclosure 2.

In certain simulations one might require an interior mesh to represent other coupled physics within the cavity area of the enclosure. In this case one must use the MESHED_ENCLOSURE command within the enclosure definition command block to exclude the meshed block (and its surface) from the enclosure radiation problem. For simulations in which enclosure radiation approaches the optically-thick limit one might consider utilizing a meshed enclosure in conjunction with the OPTICALLY_THICK thermal conductivity model 4.52.6 in lieu of the enclosure radiation problem.

Once a simulation has begun the enclosure radiation problem is solved for each enclosure in the order they appear in the input file. First the viewfactors are obtained, then followed by a radiosity solve for that enclosure. If any errors are encountered while resolving viewfactors or during a radiosity solve, the solution

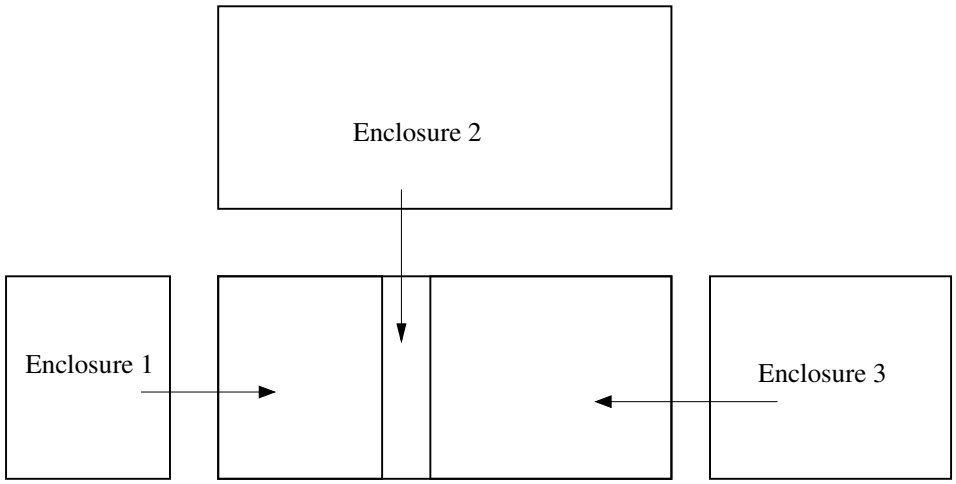


Figure 22.5. Superposed Radiation Enclosures.

is terminated with appropriate diagnostics. After all the individual enclosure radiation problems have been solved, those results are then coupled with the finite element solution. In most simulations involving enclosure radiation computation of the viewfactors represents a large portion of the overall solution time. If the same geometric model will be used for different analyses saving the the viewfactors to file and reusing them is highly recommended. Code facilities for writing the viewfactor files and for reading them in are provided specifically for this purpose.

After performing a simulation the analyst will often interpret the results by post-processing of the output data. To assist in this task the analyst is able to request output of the enclosure surface flux, q_n and irradiation G of equation 22.1 as well as the radiosity J 22.3. If these variables are of interest their output can be requested by including the names RAD_FLUX, IRRADIANCE, or RADIOSITY in the results output block described in a chapter 25. When using the banded-wavelength model the flux, irradiation and radiosity can be output on each band by prefacing the names with ENCLOSURE_EID, where EID is the enclosure index (the order index corresponding to where the enclosure definition appears in the input command file). In this case the values corresponding to each band will be output. All enclosure surface output is written as face variables in 3D and edge variables in 2D. Because solution variables for the partial enclosure, RAD_FLUX, IRRADIANCE, or RADIOSITY can have some relevance, their output can also be requested. However, unlike the variables corresponding to part of the surface mesh, partial enclosure variable output is requested from the enclosure definition command block and is written to a global variable.

22.5.1 Partial Enclosures

The concept of a partial enclosure is best demonstrated by use of a simplified view of radiative transfer system between surfaces 1 and 2 surrounded by free space as shown in Figure 22.6.

If radiative transfer occurs between the two bodies and the view factor between the two bodies is readily accessible a simplified network representation of the interaction, Figure 22.7 may suffice but here the interest lies in analyzing the system using enclosure radiation.

Recall that the enclosure view factors F_{ij} are computed as

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{cos\theta_i \cos\theta_j}{\pi r^2} \ \delta_{ij} dA_i dA_j \ .$$

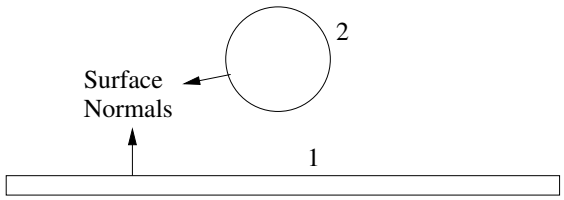


Figure 22.6. Two-Body Radiative Transfer Model

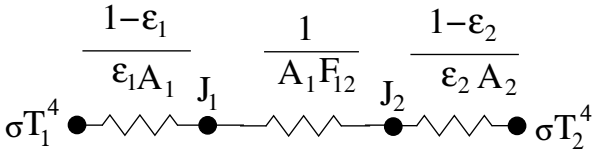


Figure 22.7. Two Surface Network Model

Determination of the view factors is a compute intensive endeavor and within most code implementations extraneous calculations are eliminated based upon the geometry. One example of this would be excluding this calculation for surfaces which are not visible to each other. Moreover, from a geometric view of enclosure surfaces, Figure 22.8, one can conclude that legitimate interactions between surfaces are those for which n_i and n_j are opposed. Thus an important feature of the enclosure model is the notion of inward facing normals. This convention effectively defines the interaction between the enclosure subfacets. A notable metric of the view factor calculation for closed surfaces (watertight enclosures) is that for each facet i the row-sum over all surface facets k, $\sum_{j=0}^{j=k} F_{ij} = 1.0$.

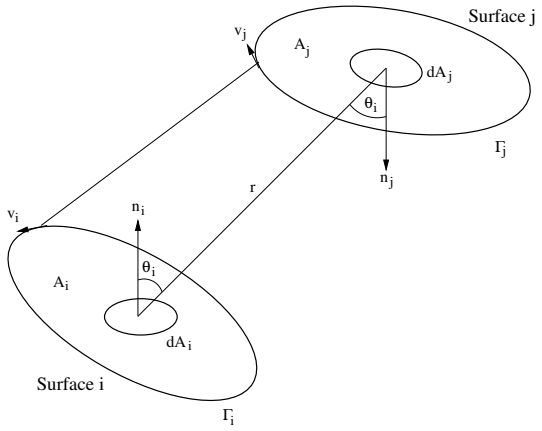


Figure 22.8. View Factor Configuration

In many cases a more complete model of the two-body configuration will include radiative transfer with the surroundings. Including these interactions requires introduction of an additional enclosure surface as shown in Figure 22.9.

Surface 3 known as the "partial enclosure" is artificially added to the numerical model (i.e. not included in the meshed discretization). Introduction of the partial enclosure is necessary in order to apply conventional

approaches for numerical evaluation of the system view factors. Here we note that algorithmically, failure to add the partial enclosure surface would be manifest in bad row-sum metrics.

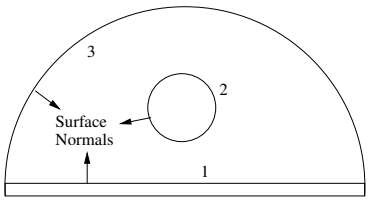


Figure 22.9. Partial Enclosure Radiative Transfer Model

Introduction of the partial enclosure leads to the thermal network representation shown in Figure 22.10 in which the partial enclosure interacts with the other bodies of the system. Since the partial enclosure is

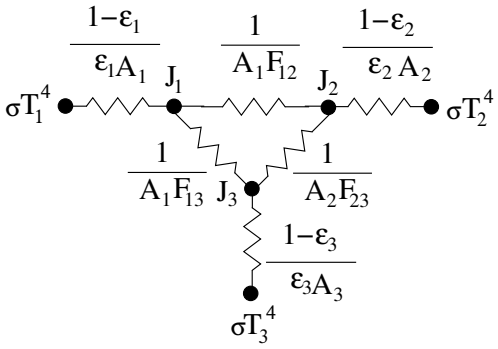


Figure 22.10. Three Surface Network Model

modeled in the same way as a real surface questions often arise concerning characterization of the partial enclosure in terms of its temperature, area and emissivity. The temperature must of course must represent the surroundings. The area should be an area that envelopes the true surfaces of the enclosures. The partial enclosure emissivity should be chosen in a manner consistent with the relationship of surface radiosity J with the emission, σT_3^4 and irradiance, G

$$J = \varepsilon\sigma T^4 - (1-\varepsilon)G \; .$$

As examples of this we consider the consequence of selecting a partial emissivity at two extremes, $\varepsilon = 0$ and $\varepsilon = 1$, both of which which reduce the network model of Figure 22.10 to that of 22.11. For $\varepsilon = 0$ from

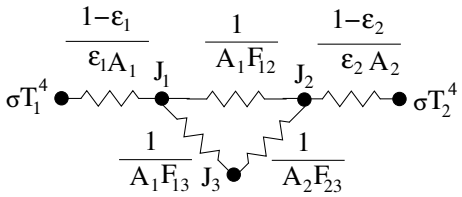


Figure 22.11. Three Surface Network Model ($\varepsilon_3 = 0$ and $\varepsilon_3 = 1$)

the network interaction between $\sigma T_3^4 - J_3$ we find that the resistance becomes infinite thus eliminating any

system interaction with T_3 as all the incident energy is reflected. This results in a modified version of the two-surface network 22.7, albeit with more attenuation. For $\varepsilon = 1$ and the network interaction between between $\sigma T_3^4 - J_3$ is eliminated. However, now the radiative transfer system sees the emissive power of the surroundings since from 22.5.1, $J = \sigma T^4$. Clearly there are no obvious choices for partial enclosure emissivity other than that it should be chosen to enable interaction of the system with the surroundings.

22.6 Viewfactor Calculation

Scope: Equation System

```
Begin Viewfactor Calculation Vf_calc
    Bsp Tree Max Depth {=|are|is} depth And Min List Length {=|are|is} l
    Compute Rule {=|are|is} VFComputeRule
    Geometric Tolerance {=|are|is} n
    Hemicube Max Subdivides {=|are|is} n
    Hemicube Min Separation {=|are|is} n
    Hemicube Resolution {=|are|is} n
    Number Of Rotations {=|are|is} n
    Output Rule {=|are|is} OutputRule
    Pairwise Monte Carlo Sample Rule {=|are|is} AMCSampleRule
    Pairwise Monte Carlo Tol1 {=|are|is} Real_value
    Pairwise Monte Carlo Tol2 {=|are|is} Real_value
    Pairwise Number Of Monte Carlo Samples {=|are|is} n
    Pairwise Number Of Visibility Samples {=|are|is} n
    Pairwise Visibility Sample Rule {=|are|is} AVSampleRule
    X-Y Plane Symmetry
    X-Z Plane Symmetry
    Y-Z Plane Symmetry
End
```

Summary This block command specifies a radiation enclosure and is used to define a method for calculating view factors. The parameter for this block corresponds to an instance of a radiation enclosure mechanics.

22.6.1 Bsp Tree Max Depth

Scope: Viewfactor Calculation

```
Bsp Tree Max Depth {=|are|is} depth And Min List Length {=|are|is} l
```

Parameter	Value	Default
depth	integer	12
l	integer	25

Summary This line command sets the BSP tree parameters.

Description This line command sets the parameters for the Binary Space Partitioning (BSP) tree. This data structure is used to accelerate the geometric operations associated with visibility to calculate the view factors. The maximum depth of the tree will set the maximum number of partitioning boxes created for a given enclosure. The minimum list length will set the desired number faces to be contained in a box. The depth of the tree will be increased until the minimum list length is met of the maximum depth is reached. It is suggested that these parameters not be adjusted from their default values.

22.6.2 Compute Rule

Scope: Viewfactor Calculation

`Compute Rule {=|are|is}` *`VFComputeRule`*

Parameter	Value	Default
VFComputeRule	`{hemicube|pairwise|read}`	`HEMICUBE`

Summary This line command sets the method for computing the view factors for this enclosure.

22.6.3 Geometric Tolerance

Scope: Viewfactor Calculation

`Geometric Tolerance {=|are|is}` *`n`*

Parameter	Value	Default
n	`real`	`1.0e-6`

Summary Set the geometric tolerance

Description This line command sets the spatial geometry tolerance for use in all geometry-related comparisons. This value should correspond to the known minimum geometric size of the element faces used to define the enclosure.

22.6.4 Hemicube Max Subdivides

Scope: Viewfactor Calculation

`Hemicube Max Subdivides {=|are|is}` *`n`*

Parameter	Value	Default
n	`integer`	2

Summary Set the upper limit of hemicube subdivides

Description This line command sets the upper limit of the number of element-face subdivides that will occur when the proximity criteria limit is exceeded (see the HEMICUBE MIN SEPARATION line command).

22.6.5 Hemicube Min Separation

Scope: Viewfactor Calculation

`Hemicube Min Separation {=|are|is}` *n*

Parameter	Value	Default
n	`real`	5

Summary — Set the hemicube minimum separation

Description — This line command sets a nondimensional minimum-separation distance allowed between element faces for the HEMICUBE method before the face is subdivided. The accuracy of the hemicube method degrades rapidly if the element faces are in close proximity to each other. This tolerance is based on dividing the normal distance between the center of the element faces by the effective diameter of the element face.

22.6.6 Hemicube Resolution

Scope: Viewfactor Calculation

`Hemicube Resolution {=|are|is}` *n*

Parameter	Value	Default
n	`integer`	400

Summary — Set the hemicube resolution

Description — This line command sets the number of uniform subpatches into which the hemicube will be divided.

22.6.7 Number Of Rotations

Scope: Viewfactor Calculation

`Number Of Rotations {=|are|is}` *n*

Parameter	Value	Default
n	`integer`	1

Summary — Set the number of internal rotations for 2D axisymmetric geometry or 3D geometry with rotation symmetry.

22.6.8 Output Rule

Scope: Viewfactor Calculation

`Output Rule {=|are|is}` *OutputRule*

Parameter	Value	Default
OutputRule	`{more verbose|none|summary|verbose}`	SUMMARY

Summary Selects the amount of information printed describing the view vactor calculation

Description By default a summary of the input values are output. The MORE VERBOSE option will print both the summary information and progress of the view factor calculation in percent done. The percent done is useful in estimating how much runtime should be allotted for the calculation.

22.6.9 Pairwise Monte Carlo Sample Rule

Scope: Viewfactor Calculation

`Pairwise Monte Carlo Sample Rule {= | are | is}` *`AMCSampleRule`*

Parameter	Value	Default
AMCSampleRule	{halton | jitter | random | uniform}	HALTON

Description This line command selects the method for the distribution of Monte Carlo integration sample points on the surface for view-factor quadrature.

22.6.10 Pairwise Monte Carlo Tol1

Scope: Viewfactor Calculation

`Pairwise Monte Carlo Tol1 {= | are | is}` *`Real_value`*

Parameter	Value	Default
Real_value	real	1.0e-5

Summary Set first of two convergence checks for Monte Carlo integration

Description This line command sets one of the two convergence checks used by the Monte Carlo integration algorithm for the PAIRWISE method. If the standard deviation of the view factor divided by the value of the view factor is less than this tolerance, the Monte Carlo integration will terminate i.e

$$\frac{\text{Std. Dev. of} F_{ij}}{F_{ij}} < tol_1$$

22.6.11 Pairwise Monte Carlo Tol2

Scope: Viewfactor Calculation

`Pairwise Monte Carlo Tol2 {= | are | is}` *`Real_value`*

Parameter	Value	Default
Real_value	real	1.0e-5

Summary Set second of two convergence checks for Monte Carlo integration

Description This line command sets one of the two convergence checks used by the Monte Carlo integration algorithm for the PAIRWISE method. If the standard deviation of the view factor is less than this tolerance, the Monte Carlo integration will terminate i.e

$$\text{Std. Dev. of} F_{ij} < tol_2$$

22.6.12 Pairwise Number Of Monte Carlo Samples

Scope: Viewfactor Calculation

`Pairwise Number Of Monte Carlo Samples {=|are|is}` *n*

Parameter	Value	Default
n	integer	1000

Summary — Set the pairwise number of Monte Carlo sample points

Description — This line command specifies the number of Monte Carlo integration sample points when Monte Carlo integration is activated during the PAIRWISE view-factor-calculation algorithm.

22.6.13 Pairwise Number Of Visibility Samples

Scope: Viewfactor Calculation

`Pairwise Number Of Visibility Samples {=|are|is}` *n*

Parameter	Value	Default
n	integer	25

Summary — Set the pairwise number visibility sample points

Description — This line command will set the number of sample points to use for evaluating visibility between element faces for the PAIRWISE view-factor-calculation algorithm. Visibility is used to select the method for calculating the view factors. If the two element faces are completely visible to each other, the view factor is calculated using Gauss quadrature or an analytic method. If the element faces are partially visible, Monte Carlo integration is used.

22.6.14 Pairwise Visibility Sample Rule

Scope: Viewfactor Calculation

`Pairwise Visibility Sample Rule {=|are|is}` *AVSampleRule*

Parameter	Value	Default
AVSampleRule	{halton|jitter|random|uniform}	UNIFORM

Summary — Set the pairwise visibility sample rule

Description — This line command selects the method for distributing the visibility sample points on the surface.

22.6.15 X-Y Plane Symmetry

Scope: Viewfactor Calculation

Summary — Specifies symmetry about the X-Y plane.

22.6.16 X-Z Plane Symmetry

Scope: Viewfactor Calculation

Summary Specifies symmetry about the X-Z plane.

22.6.17 Y-Z Plane Symmetry

Scope: Viewfactor Calculation

Summary Specifies symmetry about the Y-Z plane.

22.7 Viewfactor Smoothing

Scope: Equation System

```
Begin Viewfactor Smoothing Vf_smooth
    Convergence Tolerance {= | are | is} Param1
    Maximum Iterations {= | are | is} n
    Method {= | are | is} VFSmoothMethod
    Output Rule {= | are | is} OutputRule
    Reciprocity Rule {= | are | is} VFMatrixSymmRule
    Weight Power {= | are | is} Param1
End
```

Summary Defines a view factor smoothing scheme and its associated parameters.

Description This block command is used to define a method for smoothing the view factors. The defined method will be used for view factor smoothing only when requested with the USE VIEWFACTOR SMOOTHING line command in the ENCLOSURE DEFINITION command block.

When VIEWFACTOR SMOOTHING is requested a two step sequence is initiated

1) Reciprocity enforcement using the specified RECIPROCITY RULE
2) Rowsum enforcement as per the specified METHOD

In both steps the viewfactor matrix will be modified as prescribed.

Note
In step 1 the appropriate modifications are always applied. However, if the rowsum enforcement criteria of step 2 are not satisfied the viewfactor matrix supplied to the enclosure radiation calculation will be the viewfactor matrix resulting from step 1 and -not- the raw viewfactor matrix.

22.7.1 Convergence Tolerance

Scope: Viewfactor Smoothing

`Convergence Tolerance {=|are|is}` *Param1*

Parameter	Value	Default
Param1	`real`	`1.0e-8`

Summary — Sets convergence tolerance.

Description — This line command sets the convergence tolerance for both the SIMPLE and LEAST-SQUARES smoothing algorithms based on the following:

1) SIMPLE: $|1 - \sum F_{ij}| < tol$
2) LEAST-SQUARES: $\|r\| < tol$,

where r is the residual of the Lagrange multiplier matrix problem.

22.7.2 Maximum Iterations

Scope: Viewfactor Smoothing

`Maximum Iterations {=|are|is}` *n*

Parameter	Value	Default
n	`integer`	`500`

Summary — Set maximum iterations.

Description — This line command sets the maximum number of iterations that the SIMPLE smoothing algorithm will take.

22.7.3 Method

Scope: Viewfactor Smoothing

`Method {=|are|is}` *VFSmoothMethod*

Parameter	Value	Default
VFSmoothMethod	`{least-squares|none|simple}`	`LEAST-SQUARES`

Summary — Defines view factor smoothing method.

Description — This line command defines the algorithm for smoothing the view factors. Smoothing is a term used to describe the process of enforcing the row-sum property of the view-factor matrix given by

$$\sum F_{ij} = 1.0$$

and the reciprocity property of view factors given by

$$A_i F_{ij} = A_j F_{ji}.$$

The HEMICUBE method guarantees the row-sum property, but not necessarily the reciprocity property. The PAIRWISE method will guarantee the reciprocity property (lower diagonal is calculated and the upper triangular is filled in using reciprocity), but not necessarily the row-sum property. If these two properties are not met by the view-factor matrix, energy conservation will not be achieved.

22.7.4 Output Rule

Scope: Viewfactor Smoothing

`Output Rule {= | are | is}` ***`OutputRule`***

Parameter	Value	Default
OutputRule	`{more verbose | none | summary | verbose}`	`SUMMARY`

Summary Set the amount of output.

Description This line command sets the amount of information reported to the output screen about the view-factor smoothing.

22.7.5 Reciprocity Rule

Scope: Viewfactor Smoothing

`Reciprocity Rule {= | are | is}` ***`VFMatrixSymmRule`***

Parameter	Value	Default
VFMatrixSymmRule	`{addition | average | none | subtraction}`	`AVERAGE`

Summary Selects reciprocity enforcement rule.

Description This line command determines the method for enforcing reciprocity,

$$A_i F_{ij} = A_j F_{ji}$$

during smoothing. This command is only required for the HEMICUBE algorithm. The PAIRWISE algorithm uses reciprocity to fill in the lower triangular part of the view-factor matrix. If both of the view factors, F_{ij} and Fji, are nonzero, these factors are adjusted by averaging $A_i F_{ij}$ and $A_j F_{ji}$.

22.7.6 Weight Power

Scope: Viewfactor Smoothing

`Weight Power {= | are | is}` ***`Param1`***

Parameter	Value	Default
Param1	`real`	`2.0`

Summary Sets weight power.

Description This line command sets the power, p, used for the weights in the LEAST-SQUARES smoothing algorithm

$$w_{ij} = F^p{}_{ij}.$$

22.8 Radiosity Solver

Scope: Equation System

```
Begin Radiosity Solver Boundary condition instance name

    Convergence Tolerance {= | are | is} Tolerance

    Coupling {= | are | is} RadCouplingRule

    Maximum Iterations {= | are | is} m

    Output Rule {= | are | is} OutputRule

    Solver {= | are | is} RadSolveRule

End
```

Summary Specifies a radiation enclosure. Corresponds to an instance of radiation enclosure mechanics.

Description This block command contains methods associated with the solution of the radiosity system and the linearization method for applying the radiative heat flux to the thermal model. Note that the parameter for this block command will be used with the USE RADIOSITY SOLVER line command in the ENCLOSURE DEFINITION command block.

22.8.1 Convergence Tolerance

Scope: Radiosity Solver

```
Convergence Tolerance {= | are | is} Tolerance
```

Parameter	Value	Default
Tolerance	real	1.0e-6

Summary Sets convergence tolerance.

Description This line command sets the convergence tolerance for the iterative solution of the radiosity system. The iterative solution will stop when the ratio of the current residual to the initial residual is less than the specified tolerance:

$$\frac{\|r\|}{\|r_0\|} < tol$$

22.8.2 Coupling

Scope: Radiosity Solver

`Coupling {= | are | is}` *`RadCouplingRule`*

Parameter	Value	Default
RadCouplingRule	`{lagged | mason | smoothed}`	`MASON`

Summary Specifies linearization method.

Description This line command specifies the linearization method used to apply the radiative heat flux to the thermal model.

22.8.3 Maximum Iterations

Scope: Radiosity Solver

`Maximum Iterations {= | are | is}` *m*

Parameter	Value	Default
m	`integer`	300

Summary Sets maximum number of iterations.

Description This line command sets the maximum number of iterations allowed for the iterative solution of the radiosity equations. If the maximum number of iterations is exceeded, the calculation will fail and the thermal simulation will stop.

22.8.4 Output Rule

Scope: Radiosity Solver

`Output Rule {= | are | is}` *`OutputRule`*

Parameter	Value	Default
OutputRule	`{more verbose | none | summary | verbose}`	`NONE`

Summary Sets the information reporting level.

Description This line command will set the amount of information reported to the output screen about the radiosity solution.

22.8.5 Solver

Scope: Radiosity Solver

`Solver {= | are | is}` *`RadSolveRule`*

Parameter	Value	Default
RadSolveRule	`{chaparral cg | chaparral gmres | coupled}`	`CHAPARRAL CG`

Summary Chaparral solver selection for radiosity system.

Description This line command selects the method for solving the linear system of radiosity equations. The currently available methods are iterative linear solvers.

22.9 Enclosure Definition

Scope: Equation System

Begin Enclosure Definition *Boundary condition instance name*

Activate Mean Beam Model With Bulk Node *Model*

Add Surface *SurfaceList...*

Area Output *VariableName*

Blocking Surfaces

Dash Closure Metric Samples {=|are|is} *NumSamp* [*Tol* {=|are|is} *Value*]

Dash Solve Enclosures {=|are|is} *Values...*

Dash Solve Enclosures Union {=|are|is} *Values...*

Database Name {=|are|is} *Filename* In *VFfileFormat* Format

Disable Parallel Redistribution

Emissivity {=|are|is} *Value* [On *SurfaceName*]

Emissivity Function {=|are|is} *FunctionName* [On *SurfaceName*]

Emissivity Subroutine {=|are|is} *MySub* [On *SurfaceName*]

Emissivity Time Function {=|are|is} *FunctionName* [On *SurfaceName*]

Enable Parallel Redistribution

Input Database Name {=|are|is} *Filename*

Integrated Flux Output *VariableName*

Integrated Power Output *VariableName*

Matched Flux On *SurfaceName* {=|are|is} *FluxType* [Phase {=|are|is} *MaterialPhaseName*]

Mean Beam Length {=|are|is} *l [Parameters]...*

Meshed Enclosure Is *BlockList...*

Nonblocking Surfaces

Output Database Name {=|are|is} *Filename* In *VFfileFormat* Format

Overlapping Enclosure {=|are|is} *Bool*

Partial Enclosure Area {=|are|is} *a*

Partial Enclosure Area Subroutine {=|are|is} *FName*

Partial Enclosure Area Time Function {=|are|is} *FName*

Partial Enclosure Emissivity {=|are|is} *e*

Partial Enclosure Emissivity Subroutine {=|are|is} *FName*

Partial Enclosure Emissivity Time Function {=|are|is} *FName*

Partial Enclosure Flux Output *VariableName*

Partial Enclosure Irradiance Output *VariableName*

```
Partial Enclosure Radiosity Output VariableName
Partial Enclosure Temperature {=|are|is} t
Partial Enclosure Temperature Subroutine {=|are|is} FName
Partial Enclosure Temperature Time Function {=|are|is} FName
Preprocess Enclosures [ Options... ]
Radiosity Database Name {=|are|is} Filename
Rowsum Database Name {=|are|is} Filename
Topology Database Name {=|are|is} Filename
Use Banded Wavelength Model ModelName
Use Dash Enclosures [ type tol {=|are|is} Params... ]
Use Radiosity Solver Param0
Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList... ]
Use Viewfactor Calculation Param0
Use Viewfactor Smoothing Param0
Viewfactor Update {=|are|is} UpdateMethod [ Using Params... ]
Viewfactor Update Start Time Is Start_time
Begin Band BandName
End

End
```

Summary Specifies a radiation enclosure. Corresponds to an instance of radiation enclosure mechanics.

Description This block command is used to define an enclosure for the thermal model. There may be more than one enclosure defined for any thermal model. An enclosure definition includes the geometric aspects (list of element faces forming the surface), material properties, and setting of the algorithms for calculating the view factors between the element faces in the enclosure and the radiosity solution. The name of the enclosure is specified by the user.

** NOTE ** Enclosure names cannot begin with any sequence of numeric symbols (i.e. 0 - 9). This will cause a model to fail with very little in the way of error reporting.

22.9.1 Activate Mean Beam Model With Bulk Node

Scope: Enclosure Definition

```
Activate Mean Beam Model With Bulk Node Model
```

Parameter	Value	Default
Model	string	undefined

Summary Sets the bulk element representing the PMR material for the MBL (mean beam length) enclosure.

22.9.2 Add Surface

Scope: Enclosure Definition

`Add Surface` *`SurfaceList...`*

Parameter	Value	Default
SurfaceList	`string...`	`undefined`

Summary Adds surfaces, by name, to a boundary condition's extent.

Description This line command is used to add surfaces to the extent of a boundary condition. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

22.9.3 Area Output

Scope: Enclosure Definition

`Area Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Output the area associated with this flux boundary condition.

Description This line triggers output of the enclosure area and volume.

22.9.4 Blocking Surfaces

Scope: Enclosure Definition

Summary Specifies a blocking enclosure

Description This is the default behavior if neither this line command, nor the NONBLOCKING SURFACES line command are present. This line command informs Chaparral to consider blocking surfaces during the calculation of view factors. A blocking surface is defined as one or more element faces that occlude the view from any pair of element faces within the enclosure. If it is known that no blocking surfaces exist in the enclosure, the NONBLOCKING SURFACES line command can be included in the enclosure definition to reduce the compute time for view factors. Note that the BLOCKING SURFACES line command should be included if there is any doubt whether there are blocking surfaces in the enclosure.

22.9.5 Dash Closure Metric Samples

Scope: Enclosure Definition

`Dash Closure Metric Samples {=|are|is}` *`NumSamp`* `[` *`Tol`* `{=|are|is}` *`Value`* `]`

Parameter	Value	Default
NumSamp	integer	undefined

Summary Specify number of samples for DASH closure metric and possibly an acceptable tolerance.

Description When using DASH enclosures one may wish to assign the number of sample points used in determining closure. Optionally, one can set the acceptable value of closure metric associated with closure.

22.9.6 Dash Solve Enclosures

Scope: Enclosure Definition

`Dash Solve Enclosures {=|are|is}` *`Values...`*

Parameter	Value	Default
Values	integer...	undefined

Summary List of integer index values for Dash enclosures to perform solution. The index values must be obtained by a preprocessing step using both PREPROCESS ENCLOSURE TOPOLOGY command line at the Region level and the PREPROCESS ENCLOSURES command line in the Enclosure Definition command block.

22.9.7 Dash Solve Enclosures Union

Scope: Enclosure Definition

`Dash Solve Enclosures Union {=|are|is}` *`Values...`*

Parameter	Value	Default
Values	integer...	undefined

Summary List of integer index values for Dash enclosures to perform solution. The index values must be obtained by a preprocessing step using both PREPROCESS ENCLOSURE TOPOLOGY command line at the Region level and the PREPROCESS ENCLOSURES command line in the Enclosure Definition command block.

22.9.8 Database Name

Scope: Enclosure Definition

`Database Name {=|are|is}` *`Filename`* `In` *`VFfileFormat`* `Format`

Parameter	Value	Default
Filename	string	undefined
VFfileFormat	{ascii|binary|pnetcdf}	undefined

Summary Specifies common filename to read/write viewfactors

Description This line command specifies the database name for both input and output of the view factors for this enclosure. The file name is provided by the user, and one of three file formats is specified.

The ASCII format should be used only for small enclosures. This format is useful when the analyst wants to examine the values of the view factors that have been evaluated.

The BINARY format will produce a machine-dependent file. This view-factor file can only be read on the same machine in subsequent calculations.

Both the ASCII and BINARY file formats may only be read on the same number of processors as they were written with, and are deprecated as of Aria version 4.43.5. The replacement PNETCDF format produces a single machine-independent file that may be written or read in runs using any number of processors.

22.9.9 Disable Parallel Redistribution

Scope: Enclosure Definition

Summary Disable parallel redistribution of viewfactor matrix.

Description Ensures that the default distribution of the viewfactor matrix is distribution of rows across MPI ranks is enforced - no parallel distribution of the view factor matrix.

22.9.10 Emissivity

Scope: Enclosure Definition

`Emissivity {=|are|is}` *`Value`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
Value	`real`	`undefined`

Summary Sets a constant value of emissivity for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.9.11 Emissivity Function

Scope: Enclosure Definition

`Emissivity Function {=|are|is}` *`FunctionName`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
FunctionName	`string`	`undefined`

Summary Sets a emissivity function for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.9.12 Emissivity Subroutine

Scope: Enclosure Definition

`Emissivity Subroutine {=|are|is}` *`MySub`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
MySub	string	undefined

Summary Sets a emissivity user subroutine for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

Also, the software supports using locally scoped user data for most user subroutines, but I haven't figured out a syntax for it here yet. So it is not yet supported. If you need to get data into this subroutine, use the region's "REAL DATA" and "INTEGER DATA" line commands.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.9.13 Emissivity Time Function

Scope: Enclosure Definition

`Emissivity Time Function {=|are|is}` *`FunctionName`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
FunctionName	string	undefined

Summary Sets a emissivity function of time for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.9.14 Enable Parallel Redistribution

Scope: Enclosure Definition

Summary Enables parallel redistribution of viewfactor matrix.

Description Invokes redistribution of the view factor matrix from a distribution of matrix rows across MPI ranks to a 2D distribution of row and column blocks to achieve better performance in parallel. A current limitation of this option is that the Tpetra redistribution capability will use more memory than necessary and may not be a appropriate if one is memory constrained on a platform.

22.9.15 Input Database Name

Scope: Enclosure Definition

`Input Database Name {=|are|is}` *`Filename`*

Parameter	Value	Default
Filename	`string`	`undefined`

Summary — Specifies filename to read viewfactors from

Description — This line command provides the name of the view-factor file. If this line command is present, the file containing the given view-factor matrix will be opened and read. The calculation of view factors will not take place. Note that you must also specify a value of READ for the COMPUTE RULE command in the associated VIEWFACTOR CALCULATION command block.

22.9.16 Integrated Flux Output

Scope: Enclosure Definition

`Integrated Flux Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary — Calculate the average flux associated with this flux boundary condition.

Description — This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power. This power is then divided by the total area of the surface to obtain the average flux on the surface, and stored in a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

22.9.17 Integrated Power Output

Scope: Enclosure Definition

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary — Calculate the total power associated with this flux boundary condition.

Description — This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

22.9.18 Matched Flux On

Scope: Enclosure Definition

`Matched Flux On` *SurfaceName* `{=|are|is}` *FluxType* `[ Phase {=|are|is}` *MaterialPhaseName* `]`

Parameter	Value	Default
SurfaceName	string	undefined
FluxType	string	undefined

Summary Indicates an equivalent matching flux for surfaces within the enclosure that border element blocks which do not have the ENERGY equation defined but do have an equivalent equation (such as ENTHALPY) defined. The primary use for this is in OMD simulations and should be considered at the same reliability level as OMD.

22.9.19 Mean Beam Length

Scope: Enclosure Definition

`Mean Beam Length {=|are|is}` *l [Parameters]...*

Parameter	Value	Default
l	string	undefined
Parameters	[string]...	undefined

Summary Sets the expression for the radiant decay length L for the MBL (mean beam length) enclosure.

22.9.20 Meshed Enclosure

Scope: Enclosure Definition

`Meshed Enclosure Is` *BlockList...*

Parameter	Value	Default
BlockList	string...	undefined

Summary Adds blocks, by name, to the meshed enclosure extent.

Description In the event one wishes to solve a conduction problem within an enclosure as well as the radiation problem the interior discretization must be defined. This line command is used to add blocks to the extent of a enclosure interior. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, block 12 would be added by this line command using the surface name block_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

22.9.21 Nonblocking Surfaces

Scope: Enclosure Definition

Summary Specifies a non-blocking enclosure

Description See the BLOCKING SURFACES command for a complete description.

22.9.22 Output Database Name

Scope: Enclosure Definition

`Output Database Name {=|are|is}` *`Filename`* `In` *`VFfileFormat`* `Format`

Parameter	Value	Default
Filename	`string`	`undefined`
VFfileFormat	`{ascii|binary|pnetcdf}`	`undefined`

Summary Specify filename to write viewfactors to

Description This line command activates the output of the view factors calculated during a simulation. The file name is provided by the user, and one of three file formats is specified. See the "DATABASE NAME" line command description for more detail on the file formats.

22.9.23 Overlapping Enclosure

Scope: Enclosure Definition

`Overlapping Enclosure {=|are|is}` *`Bool`*

Parameter	Value	Default
Bool	`{false|true}`	`undefined`

Summary Designates this enclosure as overlapping other enclosures.

Description On occasion one might want to define an enclosure radiation problem as a superposition of several enclosures. In this case one must designate one of the enclosures as overlapping the others in order to distinguish it from the other enclosures. This command adds a qualifier to the enclosure definition to enable making this distinction possible.

22.9.24 Partial Enclosure Area

Scope: Enclosure Definition

`Partial Enclosure Area {=|are|is}` *`a`*

Parameter	Value	Default
a	`real`	`undefined`

Summary Constant value for the partial enclosure area associated with this enclosure radiation flux boundary condition.

22.9.25 Partial Enclosure Area Subroutine

Scope: Enclosure Definition

`Partial Enclosure Area Subroutine {=|are|is}` *`FName`*

Parameter	Value	Default
FName	`string`	`undefined`

Summary User-defined function name for the partial enclosure area associated with this enclosure radiation flux boundary condition.

22.9.26 Partial Enclosure Area Time Function

Scope: Enclosure Definition

`Partial Enclosure Area Time Function {=|are|is}` *FName*

Parameter	Value	Default
FName	`string`	`undefined`

Summary Time-dependent function name for the partial enclosure area associated with this enclosure radiation flux boundary condition.

22.9.27 Partial Enclosure Emissivity

Scope: Enclosure Definition

`Partial Enclosure Emissivity {=|are|is}` *e*

Parameter	Value	Default
e	`real`	`undefined`

Summary Constant value for the partial enclosure emissivity associated with this enclosure radiation flux boundary condition.

22.9.28 Partial Enclosure Emissivity Subroutine

Scope: Enclosure Definition

`Partial Enclosure Emissivity Subroutine {=|are|is}` *FName*

Parameter	Value	Default
FName	`string`	`undefined`

Summary User-defined function name for the partial enclosure emissivity associated with this enclosure radiation flux boundary condition.

22.9.29 Partial Enclosure Emissivity Time Function

Scope: Enclosure Definition

`Partial Enclosure Emissivity Time Function {=|are|is}` *FName*

Parameter	Value	Default
FName	`string`	`undefined`

Summary Time-dependent function name for the partial enclosure emissivity associated with this enclosure radiation flux boundary condition.

22.9.30 Partial Enclosure Flux Output

Scope: Enclosure Definition

`Partial Enclosure Flux Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Output the flux associated with the partial enclosure.

Description During the simulation the flux associated with the partial enclosure is output to the ExodusII file as a global variable.

22.9.31 Partial Enclosure Irradiance Output

Scope: Enclosure Definition

`Partial Enclosure Irradiance Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Output the irradiance associated with the partial enclosure.

Description During the simulation the irradiance associated with the partial enclosure is output to the ExodusII file as a global variable.

22.9.32 Partial Enclosure Radiosity Output

Scope: Enclosure Definition

`Partial Enclosure Radiosity Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Output the radiosity associated with the partial enclosure.

Description During the simulation the radiosity associated with the partial enclosure is output to the ExodusII file as a global variable.

22.9.33 Partial Enclosure Temperature

Scope: Enclosure Definition

`Partial Enclosure Temperature {=|are|is}` *`t`*

Parameter	Value	Default
t	`real`	`undefined`

Summary Constant value for the partial enclosure temperature associated with this enclosure radiation flux boundary condition.

22.9.34 Partial Enclosure Temperature Subroutine

Scope: Enclosure Definition

`Partial Enclosure Temperature Subroutine {=|are|is}` *FName*

Parameter	Value	Default
FName	string	undefined

Summary User-defined function name for the partial enclosure temperature associated with this enclosure radiation flux boundary condition.

22.9.35 Partial Enclosure Temperature Time Function

Scope: Enclosure Definition

`Partial Enclosure Temperature Time Function {=|are|is}` *FName*

Parameter	Value	Default
FName	string	undefined

Summary Time-dependent function name for the partial enclosure temperature associated with this enclosure radiation flux boundary condition.

22.9.36 Preprocess Enclosures

Scope: Enclosure Definition

Summary Preprocess enclosures subject to options: ORIGINAL_FACES, ENCLOSURE_FACETS, ENCLOSURE_FACET_EDGES, CONTACT_FACETS, CONTACT_FACET_EDGES and COARSE_MESH. When invoked at Region preprocessing is performed and the simulation will terminate.

Description For debug and inspection purposes the surfaces defined in the enclosure definitions will be processed, then output to the exodus database before terminating the simulation. If no options are chosen the default is to select all the available information. Options are: ORIGINAL_FACES, ENCLOSURE_FACETS, ENCLOSURE_FACET_EDGES, CONTACT_FACETS, CONTACT_FACET_EDGES and COARSE_MESH. Facets are output as triangle elements, edges as bars and faces as shell elements.

22.9.37 Radiosity Database Name

Scope: Enclosure Definition

`Radiosity Database Name {=|are|is}` *Filename*

Parameter	Value	Default
Filename	`string`	`undefined`

Summary Specify filename to write enclosure rowsum error and radiosity results to

Description This line command activates the output of the view factor rowsum error calculated during a simulation. Additionally, this file will be used to also output the radiosity and flux values during subsequent calls to the radiosity solver. The file name is provided by the user and the output format is ExodusII. This file is useful when the analyst wants to where in the enclosure the maximum errors are occurring. The rowsum error is given by

$$1 - \sum_{j=1}^{N_{facets}} F_{ij}$$

for each row i in the view-factor matrix (which corresponds to a face in the enclosure). Use of this option will negatively affect performance and is intended for use by developers.

22.9.38 Rowsum Database Name

Scope: Enclosure Definition

`Rowsum Database Name {=|are|is}` *`Filename`*

Parameter	Value	Default
Filename	`string`	`undefined`

Summary Specify filename to write enclosure rowsum error results to

Description This line command activates the output of the view factor rowsum error calculated during a simulation. The file name is provided by the user and the output format is ExodusII. This file is useful when the analyst wants to where in the enclosure the maximum errors are occurring. The rowsum error is given by

$$1 - \sum_{j=1}^{N_{facets}} F_{ij}$$

for each row i in the view-factor matrix (which corresponds to a face in the enclosure).

22.9.39 Topology Database Name

Scope: Enclosure Definition

`Topology Database Name {=|are|is}` *`Filename`*

Parameter	Value	Default
Filename	`string`	`undefined`

Summary Specify filename to write enclosure topology to

Description This line command activates the output of the enclosure topology, taking into account all specified symmetry. The file name is provided by the user and the output format is ExodusII.

22.9.40 Use Banded Wavelength Model

Scope: Enclosure Definition

`Use Banded Wavelength Model` *`ModelName`*

Parameter	Value	Default
ModelName	`string`	`undefined`

Summary — Use a specified banded wavelength model for use in enclosure radiation.

Description — Requests that a specific banded wavelength model named modelName, be associated with an enclosure, where the modelName is defined outside of the enclosure definition block. Only one BANDED WAVELENGTH MODEL can be used for an enclosure.

22.9.41 Use Dash Enclosures

Scope: Enclosure Definition

Summary — Perform DASH contact search on the enclosure surfaces.

Description — Performing a DASH contact search on the enclosure surfaces will prevent any faces in contact from being processed as part of an enclosure. Optionally one may define a relative tolerance based upon the characteristic element length, or absolute tolerances in positive and negative directions from the enclosure surface.

22.9.42 Use Radiosity Solver

Scope: Enclosure Definition

`Use Radiosity Solver` *`Param0`*

Parameter	Value	Default
Param0	`string`	`undefined`

Summary — Specifies which radiosity solver to use.

Description — This line command is used to identify the radiosity solver for use with this enclosure radiation problem. The details of the radiosity solver are defined using the RADIOSITY SOLVER command block. The parameter for this line command is the parameter used with the RADIOSITY SOLVER block command.

22.9.43 Use Toggle Block

Scope: Enclosure Definition

`Use Toggle Block` *`ToggleName`* `[ {@|at|for|in|on|over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

22.9.44 Use Viewfactor Calculation

Scope: Enclosure Definition

`Use Viewfactor Calculation` *Param0*

Parameter	Value	Default
Param0	`string`	`undefined`

Summary Specifies which view factor calculation to use.

Description This line command is used to specify the name of a method for calculating view factors defined using the VIEWFACTOR CALCULATION command block. The parameter for this line command is the parameter used to name the VIEWFACTOR CALCULATION block command.

22.9.45 Use Viewfactor Smoothing

Scope: Enclosure Definition

`Use Viewfactor Smoothing` *Param0*

Parameter	Value	Default
Param0	`string`	`undefined`

Summary Specifies which view factor smoother to use.

Description This line command is used to specify the name of a method for smoothing view factors defined using the VIEWFACTOR SMOOTHING command block. The parameter for this line command is the parameter used with the VIEWFACTOR SMOOTHING block command.

22.9.46 Viewfactor Update

Scope: Enclosure Definition

`Viewfactor Update {=|are|is}` *UpdateMethod* `[ Using` *Params...* `]`

Parameter	Value	Default
UpdateMethod	`{affected_area|continuous|freeze|interval|standard|times}`	`STANDARD`

Summary Specify the viewfactor re-compute strategy.

Description The view factor calculation oftentimes dominates the computational cost of a thermal analysis. Thus for cases in which only the geometry changes due to surface deformation and the

analyst views the change to be small enough so as to not affect the view factors greatly, one can select one of the strategies below for update of the view factors.

Known Issue: This command has only been enabled to work with models with mesh motion and will not work with standard element death.

`standard` is the default re-compute strategy, update on mesh change.

`freeze` means the view factor update is deactivated for subsequent updates. Thus the first view factor calculation determines the view factors used throughout the simulation.

`continuous` means the view factors will be computed every time step of a transient simulation.

`affected_area` is relevant only when used with Element Death. For every view factor calculation the enclosure surface area is computed and stored as a reference area. At subsequent solution steps the area corresponding to elements which are candidates for Element Death is accumulated. The radio of candidate death element area to the original area is used as the criteria of whether Element Death and a new view factor update will occur. If `affected_area` is chosen and Element Death is not present the re-compute strategy will revert to `standard`. Here the `Params` value is the ratio of affected area to total area for which element death can occur.

Known Issue: The `affected_area` option has not yet been enabled for use and will not control the viewfactor update frequency.

`interval` specifies the time interval between view factor updates. Here `Params` is a single value time increment.

`times` selects specific times at which the view factors are recalculated. Here `Params` can be several discrete time values.

For `interval` and `times`, one can also select a time at which view factor updates are considered using the `view factor update start time` command line.

22.9.47 Viewfactor Update Start Time

Scope: Enclosure Definition

`Viewfactor Update Start Time Is` *`Start_time`*

Parameter	Value	Default
Start_time	`real`	`0.0`

Summary — Specify start time for possible re-compute of viewfactors.

Description — Specify the time after which viewfactor re-compute can occur. Command is active only for UPDATE FACTOR method = TIMES or INTERVAL.

22.10 Banded Wavelength Model

Scope: Equation System

```
Begin Banded Wavelength Model ModelName
    Band BandName {=|are|is} LowValue HighValue
End
```

Summary This command block specifies the wavelength distribution for a banded wavelength model.

Description Defines a piecewise constant banded wavelength discretization model of the thermal wavelength spectrum and associate it with name modelName. The same modelName can be used for different boundary conditions.

22.10.1 Band

Scope: Banded Wavelength Model

```
Band BandName {=|are|is} LowValue HighValue
```

Parameter	Value	Default
BandName	string	undefined
LowValue	real	0.0
HighValue	real	REAL_MAX

Summary Specify the range of a wavelength band in microns μm for a surface or collection of surfaces undergoing radiative heat transfer.

Description Defines the upper and lower limits of a wavelength band in microns μm. Upper and lower limit values can intersect those of adjacent bands but cannot overlap into previously a defined BAND.

22.11 Band

Scope: Enclosure Definition

```
Begin Band BandName
    Emissivity {=|are|is} Value [ On SurfaceName ]
    Emissivity Function {=|are|is} FunctionName [ On SurfaceName ]
    Emissivity Subroutine {=|are|is} MySub [ On SurfaceName ]
    Emissivity Time Function {=|are|is} FunctionName [ On SurfaceName ]
End
```

Summary This nested command block specifies the emissivity distribution for the surfaces in the enclosure for the given band.

Description Defines the emissivity distribution for the surfaces and/or default values for the band with name bandName.

22.11.1 Emissivity

Scope: Band

`Emissivity {=|are|is}` *`Value`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
Value	`real`	`undefined`

Summary

Sets a constant value of emissivity for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.11.2 Emissivity Function

Scope: Band

`Emissivity Function {=|are|is}` *`FunctionName`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
FunctionName	`string`	`undefined`

Summary

Sets a emissivity function for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.11.3 Emissivity Subroutine

Scope: Band

`Emissivity Subroutine {=|are|is}` *`MySub`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
MySub	`string`	`undefined`

Summary

Sets a emissivity user subroutine for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

Also, the software supports using locally scoped user data for most user subroutines, but I haven't figured out a syntax for it here yet. So it is not yet supported. If you need to get data into this subroutine, use the region's "REAL DATA" and "INTEGER DATA" line commands.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.11.4 Emissivity Time Function

Scope: Band

`Emissivity Time Function {=|are|is}` *`FunctionName`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
FunctionName	string	undefined

Summary — Sets a emissivity function of time for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

22.11.5 Dash Closure Metric Samples

Scope:

`Dash Closure Metric Samples {=|are|is}` *`NumSamp`* `[` *`Tol`* `{=|are|is}` *`Value`* `]`

Parameter	Value	Default
NumSamp	integer	undefined

Summary — Specify number of samples for DASH closure metric and possibly an acceptable tolerance.

Description — When using DASH enclosures one may wish to assign the number of sample points used in determining closure. Optionally, one can set the acceptable value of closure metric associated with closure.

22.11.6 Dash Closure Metric Samples

Scope:

`Dash Closure Metric Samples {=|are|is}` *`NumSamp`* `[` *`Tol`* `{=|are|is}` *`Value`* `]`

Parameter	Value	Default
NumSamp	integer	undefined

Summary — Specify number of samples for DASH closure metric and possibly an acceptable tolerance.

Description — When using DASH enclosures one may wish to assign the number of sample points used in determining closure. Optionally, one can set the acceptable value of closure metric associated with closure.

22.11.7 Preprocess Enclosures

Scope:

Summary Preprocess enclosures subject to options: ORIGINAL_FACES, ENCLOSURE_FACETS, ENCLOSURE_FACET_EDGES, CONTACT_FACETS, CONTACT_FACET_EDGES and COARSE_MESH. When invoked at Region preprocessing is performed and the simulation will terminate.

Description For debug and inspection purposes the surfaces defined in the enclosure definitions will be processed, then output to the exodus database before terminating the simulation. If no options are chosen the default is to select all the available information. Options are: ORIGINAL_FACES, ENCLOSURE_FACETS, ENCLOSURE_FACET_EDGES, CONTACT_FACETS, CONTACT_FACET_EDGES and COARSE_MESH. Facets are output as triangle elements, edges as bars and faces as shell elements.

22.11.8 Use Dash Enclosures

Scope:

Summary Perform DASH contact search on the enclosure surfaces.

Description Performing a DASH contact search on the enclosure surfaces will prevent any faces in contact from being processed as part of an enclosure. Optionally one may define a relative tolerance based upon the characteristic element length, or absolute tolerances in positive and negative directions from the enclosure surface.

Chapter 23

SPn Radiation Reference

23.1 Radiative Transport

Radiative transport, which is another mode of heat transfer, describes the spatial variation of radiative intensity corresponding to a given direction and at a given wavelength within a radiatively participating medium. This is governed by the Boltzmann Equation or Radiative Transport Equation (RTE) which represents a balance between absorption, emission and scattering. For the purpose of thermal analysis, we make certain assumptions that simplify the RTE. These are

1. Gray Media: The optical properties of the material medium are not a function of wavelength i.e no spectral banding.
2. Steady State: Energy transport is propagated by photons which travel at the speed of light. This means that there are no transient effects relative to the time scale of other physics.
3. Isotropic Scattering: Since the scattering phase function is rarely known, we simply assume isotropic scattering where the probability of intensity being propagated in a given direction has equal probability of being scattered in another direction.

Given the above assumptions, we may write the RTE as

$$\Omega \cdot \boldsymbol{\nabla} I(\Omega) + (\sigma_A + \sigma_S) I(\Omega) = \sigma_A I_b + \frac{\sigma_S}{4\pi} G, \tag{23.1}$$

where σ_A is the absorption coefficient, σ_S is the scattering coefficient, $I(\Omega)$ is the intensity along the direction Ω, T is the temperature and the angle-integrated intensity is $G = \int_{4\pi} I(\Omega)\, d\Omega$. The black body radiation, I_b is defined by $\frac{\sigma T^4}{\pi}$. Note that for situations in which the scattering coefficient is zero, the RTE reduces to a set of linear, decoupled equations for each intensity to be solved. Discrete Ordinate (DO) methods such as S_N have been developed to help solve this set of coupled equations by selecting a set of prescribed directions Ω_k, solving for the discrete intensities I_k and combining these intensities to represent the radiative field. These methods suffer from high computational cost due to the fact that a large number of ordinate directions is required to adequately resolve the intensity field and avoid "ray effects". For a given quadrature order N, the number of ordinate directions scales as N^2.

23.1.1 Simplified Spherical Harmonics (SP_N) Equations

Beta Capability: SP_N is not well tested and should not be considered a production capability

The simplified spherical harmonics (SP_N) approximation to the radiative transport equation (RTE) was first proposed by Gelbard for reactor analysis in the early 1960s. The initial derivation involved replacing

the spatial derivatives in the 1D spherical harmonics (P_N) approximation with their 3D analogs. It was later shown that a more rigorous theoretical basis was possible and that the SP_N equations may be derived as either an asymptotic correction to the diffusion limit or from the use of certain trial functions in the self-adjoint variational characterization of the even-parity form of the RTE. There are a number of equivalent forms for the SP_N equations in the literature. We choose to use the "canonical" form which is derived from the 1D even-parity discrete ordinates equations.

$$-\boldsymbol{\nabla}\cdot\left(\frac{\mu_n^2}{\sigma_T}\boldsymbol{\nabla}I_n\right)+\sigma_T I_n = 4\pi\sigma_S\sum_{m=1}^{\frac{N+1}{2}} w_m I_m + \sigma_A I_b \tag{23.2}$$

where

- σ_T is the extinction coefficient, $\sigma_T = \sigma_A + \sigma_S$
- μ_n is the nth quadrature point in a (N+1)-point Gauss set on [-1, 1]
- I_n is the angular intensity at quadrature point n
- w_n is the n^{th} quadrature weight

The appropriate boundary conditions for the "canonical" SPn equations are derived from the boundary conditions for the 1D even-parity discrete ordinates equations. We consider a Mark boundary condition of the 1st order intensity BC

$$I_n = \epsilon I_b + (1-\epsilon)\sum_{\vec{n}\cdot\vec{\Omega}_j<0} I_j w_j \left|\vec{n}\cdot\vec{\Omega}_j\right| \tag{23.3}$$

such that in canonical form, it yields

$$-\frac{\mu_n}{\sigma_T}\boldsymbol{\nabla}I_n\cdot\vec{n} = \frac{\epsilon}{2-\epsilon}\left(I_n - I_b\right) + \frac{1-\epsilon}{2-\epsilon}\left[\frac{\sum_k\left(I_k - \frac{\mu_k}{\sigma_T}\boldsymbol{\nabla}I_k\cdot\vec{n}\right)\mu_k w_k}{\sum\mu_k w_k} - I_n + \frac{\mu_n}{\sigma_T}\boldsymbol{\nabla}I_n\cdot\vec{n}\right] \tag{23.4}$$

where

- ϵ is the emissivity
- $\vec{n}$ is the surface normal

The SP_N equations are solved for the unknown intensities to provide an approximate solution to the RTE. The angle-integrated intensity is then found and used in the source term for the material energy equation. It is to be noted from the form of the SP_N equations that this cannot be used in vacuum or near-vacuum situations where σ_T 0. Also analysis by Zheng et al shows that the existence of a unique solution may be proved when absorption effects are non-negligible and the geometry is small.

Chapter 24

Contact Reference

24.1 Contact Overview

The term "contact" refers to a situation where a simulation needs to be performed that spans a non-contiguously meshed interface, i.e. a nonconformal interface. One example of a nonconformal interface is one that has adjacent blocks and there appears to be a single node at a certain location in space on the interfaces between blocks, but in fact there are distinct nodes that have identical, or nearly identical coordinates. In other words, the mesh nodes have not been merged or equivalenced. More generally, a nonconformal interface does not require that mesh lines match at the interface between block boundaries.

There are two modeling situations that lead to the use of nonconformal interfaces in thermal analysis. The first occurs when compatibility conditions are enforced at the interface to require that the temperature be continuous across the interface. This situation arises as a convenient way to generate the mesh for particularly complex geometries since both parts can be meshed independently. The second modeling situation that occurs is not a matter of convenience, but of physics. It may be that two parts are known to fit imperfectly, so that there is a gap between the two parts that has a certain known resistance. In this case, the enforcement is known as gap conductance (or resistance) and enforces a discontinuous solution across the gap by modeling the energy flux that is "lost" across the gap.

In Aria, all contact is implemented via the nonconformal contact algorithm, and various enforcement strategies exist within this algorithm (see section on enforcement below). The enforcement strategies are similar to those that existed via the legacy generalized contact.

Warning: Nonconformal contact is now the default and recommended behavior for contact within aria. The legacy contact methods will be supported through release 4.50. Documentation for legacy versions of contact can be found in previous versions of the user manual.

The nonconformal contact method uses stk search and the dash library to identify surfaces in proximity to one another. The enforcement strategy is based on a discontinuous Galerkin approach that expresses the interaction as two surface integrals along the contact interface. The exact enforcement form depends on the chosen enforcement model. For a thermal problem with `TIED_TEMPERATURE` enforcement, one term in the surface integral penalizes the jump in the temperature weighted by an appropriate average surface conductance coefficient. The second term in the surface integral imposes the energy flux at the surface boundary, which is taken as the average energy flux on both surfaces as to be weakly conservative. If the mesh aligns perfectly at the interface such that the quadrature points of the two facets match, i.e., no interpolation is needed, this method is also discretely conservative. Errors in conservation will exist in the general case due to the discrete nature approximating the integral, although this error should decrease with the same order as the underlying numerical method. Details of the contact specification and relevant options are in subsequent sections below.

Most common cases model contact between surfaces of standard volume elements (2D quadrilateral, 2D

triangle, 3D tetrahedron or 3D hexahedron). When considering contact of shell elements with nonconformal interfaces, it is assumed that the shell nodes are coincident with nodes of a standard volume element on one side of the contact under consideration.

Warning: Shell support is limited in nonconformal contact. Specifically, contact along the edge of a shell element is not supported, i.e., a shell element that is perpendicular to another surface does not transmit a flux through the singular axis of the shell.

For a more detailed discussion of the mathematical basis of the nonconformal interface and enforcement strategies, please see the Aria Theory Manual.

24.1.1 Search and Contact Tolerance Calculation

The enforcement selected only acts upon surfaces that are defined to be in contact. Whether or not surfaces are in contact is determined via a normal tolerance that is either 1) set within the search options block of the contact definition (see syntax reference following this section), or 2) set via an automatic tolerancing operation. This normal tolerance is used on a per-integration-point projection to the closest surface, as shown in Figure 24.1.

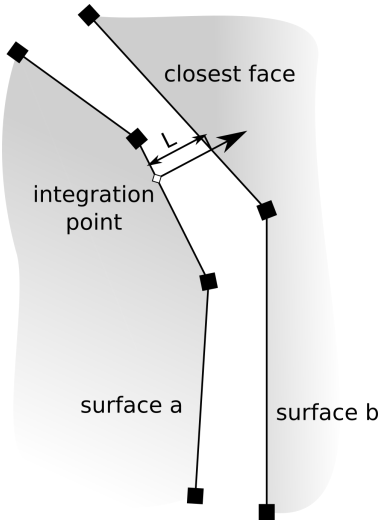


Figure 24.1. Calculation of contact distance per integration point.

In more detail: For a given surface that may be in contact, the enforcement strategy is expressed in terms of a surface integral on both surfaces (surface a and surface b in Figure 24.1). For evaluating the enforcement kernel on surface a, the surface facets are iterated, as are the integration points within each facet. The closest surface is found, as defined by the smallest distance, L, from the integration point to the surface intersection along the normal direction (L in Figure 24.1). If $L < tol$, where tol is the contact normal tolerance, then that integration point is in contact and the enforcement kernel is evaluated. If no global or interaction specific contact normal tolerance is specified, the automatically calculated contact normal tolerance is used, which is calculated as the minimum elemental characteristic length over all surface elements in a given contact surface. The elemental characteristic length is defined as the minimum length between any two nodes on the surface element.

24.1.2 Contact Enforcement

Contact enforcement strategies are discussed in detail in the following enforcement sections; however, the Thermal-Fluids Team would like to provide guidance as to the thermal enforcement strategies with the highest pedigree of code testing, verification, and validation. For cases where there is no contact resistance, i.e., the physical temperature field is contiguous, the team recommends using the `TIED_TEMPERATURE` enforcement strategy. For cases with imperfect contact that can be modeled according to a contact resistance scheme, the team recommends using the `GAP_CONDUCTANCE` enforcement strategy.

24.2 Contact Definition

Scope: Equation System

```
Begin Contact Definition Contact_name

    Contact Surface Surface_name [ Contains List_of_instances...  ]
    Opposed Normal Threshold Angle {=|are|is} Angle
    Output Rule {=|are|is} OutputRule
    Search {=|are|is} Type
    Skin All Blocks {=|are|is} Option [ Exclude Blocks_to_exlude...  ]
    Update Search Every n Steps
    Visualize Contact
    Begin Enforcement Enforcement_name
    End

    Begin Interaction Interaction_name
    End

    Begin Interaction Defaults
    End

    Begin Search Options Search_options_name
    End

End
```

Summary — Contains the commands needed to define contact for an analysis.

Description — This block command defines a scenario for the contact model. The contact surfaces are defined, the interactions are set, and the type of contact enforcement is defined.

24.2.1 Contact Surface

Scope: Contact Definition

```
Contact Surface Surface_name [ Contains List_of_instances...  ]
```

Parameter	Value	Default
Surface_name	string	undefined

Summary Defines a surface made up of a set of surfaces in the mesh file to consider for contact. Specification of an element block implies that block skinning of the element block is to take place. The specific implementation of block skinning depends on the application.

24.2.2 Opposed Normal Threshold Angle

Scope: Contact Definition

`Opposed Normal Threshold Angle {= | are | is}` *`Angle`*

Parameter	Value	Default
Angle	`real`	`90.0`

Summary Assign the angle, in degrees, for which surface normals are considered to be opposed.

Description Prescribed angle must be greater or equal to 90 degrees.

Warning: This option is not recommended for general use but serves as a temporary patch that can be used for some problems.

24.2.3 Output Rule

Scope: Contact Definition

`Output Rule {= | are | is}` *`OutputRule`*

Parameter	Value	Default
OutputRule	`{more verbose | none | summary | verbose}`	`NONE`

Summary Toggle amount of information output from the search.

24.2.4 Search

Scope: Contact Definition

Summary Specify contact search algorithm. ACME-use acme node/face contact library DASH-use dash face/face contact methods inside of Presto. ARS-use ars node/face contact methods inside Adagio/Presto.

24.2.5 Skin All Blocks

Scope: Contact Definition

Summary Informs the contact algorithms to skin all element blocks and then use the resulting surfaces in contact computations.

24.2.6 Update Search Every

Scope: Contact Definition

`Update Search Every` *n* `Steps`

Parameter	Value	Default
n	integer	undefined

Summary — Force an updating of the contact search every N steps.

Description — This command is useful if, for example, Aria is coupled to Adagio and the likelihood is strong that the mesh coordinates have changed significantly. Because contact surfaces may also appear in the definition of boundary conditions the application must selectively apply contact relations and boundary conditions appropriately. These issues are resolved automatically within the application code using the contact surface definitions and internal data masks.

24.2.7 Visualize Contact

Scope: Contact Definition

Summary — Output contact surfaces with facets marked with 1 where contact is applied and 0 otherwise.

Description — Enables output of surfaces where contact is active. Surface facets will be marked by whether contact constraints are applied on the facet - 1 or 0 otherwise.

Note: Visualization is enabled by default for contact defined using surface sidesets.

Warning: Use of this option is primarily for debugging purposes as the Results Output and History file will contain a skinned version of the contact element blocks. This may constitute a a large amount of information for discontiguously meshed models.

24.3 Enforcement

Scope: Contact Definition

`Begin Enforcement` *Enforcement_name*

`Conductance Coefficient {=|are|is}` *Coeff*

`Conductance Coefficient Contact Pressure Function {=|are|is}` *FName* `Using` *VarName*

`Conductance Coefficient For` *EquationName* `{=|are|is}` *Model* `[` *ModelParams...* `]`

`Conductance Coefficient Fortran Subroutine {=|are|is}` *Name*

`Conductance Coefficient Subroutine {=|are|is}` *FName*

`Conductance Coefficient Temperature Function {=|are|is}` *FName*

`Conductance Coefficient Time Function {=|are|is}` *FName*

`Conductance Coefficient Voltage Function {=|are|is}` *FName*

`Dash Penalty Factor {=|are|is}` *value*

`Enforcement For` *EquationName* `{=|are|is}` *EnforcementModel* `[` *ModelParams...* `]`

```
    Integer Data Values...
    Real Data Values...
    Slave Variable Slave_field To Master_field
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList... ]
End
```

Summary Contains the commands needed to define contact enforcement for an analysis in Aria.

Description This block command is used to define the type of contact between the surfaces. The name of the enforcement is specified by the user. When the contact includes some contact resistance, the conductance coefficient for the contact between the two surfaces must be specified.

24.3.1 Conductance Coefficient

Scope: Enforcement

```
Conductance Coefficient {=|are|is} Coeff
```

Parameter	Value	Default
Coeff	`real`	`undefined`

Summary Set the constant valued contact conductance coefficient used by the GAP_CONDUCTANCE and CONDUCTANCE contact models.

Description This line command defines the value of the effective heat-transfer coefficient, h_c, for the interface condition between the two surfaces. The temperature drop across the interface, due to the contact resistance, is modeled using

$$q_i n_i|_{\Gamma_{m_1}} = h_c(T_{\Gamma_{m_1}} - T_{\Gamma_{m_2}}) = q_i n_i|_{\Gamma_{m_2}}$$

where the contact coefficient or contact conductance,$h_c(x_i, t, T_{\Gamma_{m_1}}, T_{\Gamma_{m_2}})$, is modeled using a correlation that is dependent upon the two surface temperatures, location, and interface conditions. Note that smaller values of CONDUCTANCE COEFFICIENT correspond to resistance contact while larger values tend toward tied contact.

24.3.2 Conductance Coefficient Contact Pressure Function

Scope: Enforcement

```
Conductance Coefficient Contact Pressure Function {=|are|is} FName Using VarName
```

Parameter	Value	Default
FName	`string`	`undefined`
VarName	`string`	`undefined`

Summary Gap conductance coefficient from a user function in VarName

Description See CONDUCTANCE COEFFICIENT line command.

24.3.3 Conductance Coefficient For

Scope: Enforcement

`Conductance Coefficient For` *EquationName* {`=`|`are`|`is`} *Model* [*ModelParams...*]

Parameter	Value	Default
EquationName	string	undefined
Model	string	undefined

Summary — Choose the model for the conductance coefficient for the specified equation to be used in generalized contact.

Description — Choose the model for the conductance coefficient for the specified equation to be used in generalized contact.

24.3.4 Conductance Coefficient Fortran Subroutine

Scope: Enforcement

`Conductance Coefficient Fortran Subroutine` {`=`|`are`|`is`} *Name*

Parameter	Value	Default
Name	string	undefined

Summary — Specify the name of a FORTRAN user subroutine that will be used to calculate the conductance coefficient.

24.3.5 Conductance Coefficient Subroutine

Scope: Enforcement

`Conductance Coefficient Subroutine` {`=`|`are`|`is`} *FName*

Parameter	Value	Default
FName	string	undefined

Summary — Set the user defined contact conductance coefficient. See CONDUCTANCE COEFFICIENT line command for details.

Description — The subroutine must satisfy the element signature. The subroutine is passed a pair of faces that are in contact, and is expected to populate a single value for the coefficient. The temperature at the current integration point and its projection onto the face across the interface is provided. The contact interface is processed twice–once from the left side and once from the right. **Note that to ensure conservation, the subroutine should calculate the same value of the gap conductance on both sides of the interface.**

24.3.6 Conductance Coefficient Temperature Function

Scope: Enforcement

`Conductance Coefficient Temperature Function {=|are|is}` *FName*

Parameter	Value	Default
FName	string	undefined

Summary Set the temperature dependent contact conductance coefficient.

Description See CONDUCTANCE COEFFICIENT line command.

24.3.7 Conductance Coefficient Time Function

Scope: Enforcement

`Conductance Coefficient Time Function {=|are|is}` *FName*

Parameter	Value	Default
FName	string	undefined

Summary Set the time dependent contact conductance coefficient.

Description See CONDUCTANCE COEFFICIENT line command.

24.3.8 Conductance Coefficient Voltage Function

Scope: Enforcement

`Conductance Coefficient Voltage Function {=|are|is}` *FName*

Parameter	Value	Default
FName	string	undefined

Summary Set the temperature dependent contact conductance coefficient.

Description See CONDUCTANCE COEFFICIENT line command.

24.3.9 Dash Penalty Factor

Scope: Enforcement

`Dash Penalty Factor {=|are|is}` *value*

Parameter	Value	Default
value	real	undefined

Summary Defines a penalty factor to use for the dash constraints

24.3.10 Enforcement For

Scope: Enforcement

`Enforcement For` *EquationName* {= | are | is} *EnforcementModel* [*ModelParams...*]

Parameter	Value	Default
EquationName	string	undefined
EnforcementModel	string	undefined

Summary

Defines the model for enforcement for the command block interactions, either tied or resistance/conductance contact.Tied contact can be of type generalized or Lagrange multiplier/MPC, where generalized contact is the preferred contact algorithm. 24.4

Description

This command line sets the type of contact problem for interactions defined in the current CONTACT DEFINITION command block and the equation to which the enforcement applies.

The Generalized contact enforcement model can be either GAP_CONDUCTANCE, CONDUCTANCE or TIED_DOF. In the case of resistance contact one must define a CONDUCTANCE COEFFICIENT model. To specify that TIED contact will be enforced for DOF my_DOF_name, the enforcement model will be TIED_my_DOF_name, where the solution DOF for Equationame is defined in the EQ command line.

Lagrange multiplier enforcement is specified simply as model TIED and must be accompanied by a SLAVE VARIABLE command line within the ENFORCEMENT command block. MPC contact requires that one supply MASTER and SLAVE surface definitions in the INTERACTION command block. Additionally one should consider using the MATRIX REDUCTION = FEI-REMOVE-SLAVES in the equation solver command block for MPC constraint reduction.

DASH contact enforcement will be enforced only in a TIED sense.

24.3.11 Integer Data

Scope: Enforcement

`Integer Data` *Values...*

Parameter	Value	Default
Values	integer...	undefined

Summary

List of integer data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

24.3.12 Real Data

Scope: Enforcement

`Real Data` *Values...*

Parameter	Value	Default
Values	real...	undefined

Summary List of real data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

24.3.13 Slave Variable

Scope: Enforcement

`Slave Variable` *`Slave_field`* `To` *`Master_field`*

Parameter	Value	Default
Slave_field	string	undefined
Master_field	string	undefined

Summary Defines the variables involved in enforcement for this interaction.

Description This line command sets the master and slave variables involved in the current tied contact definition.

24.3.14 Use Toggle Block

Scope: Enforcement

`Use Toggle Block` *`ToggleName`* `[ {@|at|for|in|on|over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	string	undefined

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

24.4 Enforcement Models

24.4.1 Enforcement For Continuity = Tied Ip Continuity

Scope: Enforcement

24.4.2 Enforcement For Current = Bulter Volmer

Scope: Enforcement

`Enforcement For Current = Bulter Volmer [ Toggle =` *`toggle`* `|F =` *`f`* `|R =` *`r`* `|Alpha_A =` *`alpha_a`* `|Alpha_C =` *`alpha_c`* `]`

Parameter	Value	Default
toggle	"string"	undefined
f	real	undefined
r	real	undefined
alpha_a	real	undefined
alpha_c	real	undefined

24.4.3 Enforcement For Current = Conductance

Scope: Enforcement

```
Enforcement For Current = Conductance [ Toggle = toggle ]
```

Parameter	Value	Default
toggle	"string"	undefined

Summary Applies a contact current flux of the form $C(V_1 - V_2)/2$ to sides 1 and 2 of the contact interface where C is a user specified conductance coefficient. Used in conjunction with a CONDUCTANCE COEFFICIENT model.

24.4.4 Enforcement For Current = Gap Conductance

Scope: Enforcement

```
Enforcement For Current = Gap Conductance [ Toggle = toggle ]
```

Parameter	Value	Default
toggle	"string"	undefined

Summary Applies a contact current flux of the form $\tilde{C}(V_1 - V_2)/2$ to sides 1 and 2 of the contact interface. With C, a user defined coefficient and $\bar{C}$, an average conductance coefficient, then $\tilde{C} = C$ for $C \ll \bar{C}$ and $\bar{C}$ otherwise. Used in conjunction with a CONDUCTANCE COEFFICIENT model.

24.4.5 Enforcement For Current = Tied Voltage

Scope: Enforcement

```
Enforcement For Current = Tied Voltage [ Toggle = toggle |Co_Field = co_field ]
```

Parameter	Value	Default
toggle	"string"	undefined
co_field	integer	undefined

Summary Applies a contact current flux of the form $(\sigma_1 \nabla V_1 \cdot n_1 + \sigma_2 \nabla V_2 \cdot n_2 + \bar{C}(V_1 - V_2))/2$ to sides 1 and 2 of the contact interface where σ is the electrical conductivity and $\bar{C}$ is an average conductance coefficient.

24.4.6 Enforcement For Cvfem Continuity = Tied Continuity

Scope: Enforcement

24.4.7 Enforcement For Cvfem Energy = Conductance

Scope: Enforcement

`Enforcement For Cvfem Energy = Conductance [ Toggle =` *`toggle`* `|Bulk_Node =` *`bulk_node`* `]`

Parameter	Value	Default
toggle	"string"	undefined
bulk_node	"string"	undefined

Summary Applies a contact heat flux of the form $C(T_1 - T_2)/2$ to sides 1 and 2 of the contact interface where C is a user specified conductance coefficient. Used in conjunction with a CONDUCTANCE COEFFICIENT model.

24.4.8 Enforcement For Cvfem Energy = Gap Conductance

Scope: Enforcement

`Enforcement For Cvfem Energy = Gap Conductance [ Toggle =` *`toggle`* `]`

Parameter	Value	Default
toggle	"string"	undefined

Summary Applies a contact heat flux of the form $\tilde{h}(T_1 - T_2)/2$ to sides 1 and 2 of the contact interface. With h, the user defined coefficient and $\bar{h}$, an average conductance coefficient, then $\tilde{h} = h$ for $h \ll \bar{h}$ and $\bar{h}$ otherwise. Used in conjunction with a CONDUCTANCE COEFFICIENT model.

24.4.9 Enforcement For Cvfem Energy = Tied Temperature

Scope: Enforcement

`Enforcement For Cvfem Energy = Tied Temperature [ Toggle =` *`toggle`* `|Co_Field =` *`co_field`* `]`

Parameter	Value	Default
toggle	"string"	undefined
co_field	integer	undefined

Summary Applies a contact heat flux of the form $(k_1 \nabla T_1 \cdot n_2 + k_2 \nabla T_2 \cdot n_2 + \bar{h}(T_1 - T_2))/2$ to sides 1 and 2 of the contact interface where k is the thermal conductivity and $\bar{h}$ is an average conductance coefficient.

24.4.10 Enforcement For Cvfem Lumped Projection = Tied Gradp

Scope: Enforcement

24.4.11 Enforcement For Cvfem Momentum = Tied Momentum

Scope: Enforcement

24.4.12 Enforcement For Cvfem Projection = Tied Gradp

Scope: Enforcement

24.4.13 Enforcement For Cvfem Specific Dissipation Rate = Tied Sdr

Scope: Enforcement

Summary Tied_Sdr contact flux; CVFEM pressure stabilization included

24.4.14 Enforcement For Cvfem Turbulent Kinetic Energy = Tied Tke

Scope: Enforcement

Summary Tied_Tke contact flux; CVFEM pressure stabilization included

24.4.15 Enforcement For Energy = Cht Robin

Scope: Enforcement

`Enforcement For Energy = Cht Robin [ Toggle =` *`toggle`* `|Opposite_Material_Phase =` *`opposite_material_ph`*
`|Opposite_Flux_Name =` *`opposite_flux_name`* `|Coeff_Scaling =` *`coeff_scaling`* `]`

Parameter	Value	Default
toggle	"string"	undefined
opposite_material_phase	"string"	undefined
opposite_flux_name	"string"	undefined
coeff_scaling	real	undefined

Summary Applies a Robin-style BC for the energy equation: $F_{local} = F_{bc} + \alpha(T_{local} - T_{bc})$.

24.4.16 Enforcement For Energy = Conductance

Scope: Enforcement

`Enforcement For Energy = Conductance [ Toggle =` *`toggle`* `]`

Parameter	Value	Default
toggle	"string"	undefined

Summary Applies a contact heat flux of the form $C(T_1 - T_2)/2$ to sides 1 and 2 of the contact interface where C is a user specified conductance coefficient. Used in conjunction with a CONDUCTANCE COEFFICIENT model.

24.4.17 Enforcement For Energy = Conductance Salgon

Scope: Enforcement

```
Enforcement For Energy = Conductance Salgon [ Toggle = toggle |A = a |B = b |Xi = xi ]
```

Parameter	Value	Default
toggle	"string"	undefined
a	real	undefined
b	real	undefined
xi	real	undefined

Summary Applies a contact heat flux of the form $(2\hat{k}a)/(\pi b^2(1-1,14\xi))(T_1-T_2)$ to sides 1 and 2 of the contact interface where k is the thermal conductivity, $\hat{k}=(k_1+k_2)/(k_1k_2/)$ and ξ is a user specified parameter (Salgon,1997).

24.4.18 Enforcement For Energy = Contact Resistance

Scope: Enforcement

```
Enforcement For Energy = Contact Resistance [ Toggle = toggle |Opposite_Material_Phase
= opposite_material_phase ]
```

Parameter	Value	Default
toggle	"string"	undefined
opposite_material_phase	"string"	undefined

Summary Applies a contact heat flux of the form $h(T-T_opp)$ where h is the gap conductance coefficient defined for the surface material and the material phase of T_opp is optionally specified.

24.4.19 Enforcement For Energy = Gap Conductance

Scope: Enforcement

```
Enforcement For Energy = Gap Conductance [ Toggle = toggle ]
```

Parameter	Value	Default
toggle	"string"	undefined

Summary Applies a contact heat flux of the form $\tilde{h}(T_1-T_2)/2$ to sides 1 and 2 of the contact interface. With h, a user defined coefficient and $\bar{h}$, an average conductance coefficient, then $\tilde{h}=h$ for $h \ll \bar{h}$ and $\bar{h}$ otherwise. Used in conjunction with a CONDUCTANCE COEFFICIENT model.

24.4.20 Enforcement For Energy = Nitche Tied

Scope: Enforcement

```
Enforcement For Energy = Nitche Tied [ Toggle = toggle |Co_Field = co_field |Multiplier
= multiplier ]
```

Parameter	Value	Default
toggle	"string"	undefined
co_field	integer	undefined
multiplier	real	undefined

Summary Applies a contact heat flux as described in "A robust NitscheâĂŹs formulation for interface problems" by Annavarapu et al. 2012.

24.4.21 Enforcement For Energy = Phase Change

Scope: Enforcement

`Enforcement For Energy = Phase Change [ Toggle =` *`toggle`* `]`

Parameter	Value	Default
toggle	`"string"`	`undefined`

Summary Applies a energy flux to an interface based on phase change, as $q = v\rho_s(T_s C_{p,s} - T_g C_{p,g} + L)$ to sides s and g of the contact interface where v is the velocity ρ is the interface density C_p is the specific heat and L is the latent heat of fusion.

24.4.22 Enforcement For Energy = Tied D Style Rad Temperature

Scope: Enforcement

`Enforcement For Energy = Tied D Style Rad Temperature [ Toggle =` *`toggle`* `|Lambda_Scaling = lambda_scaling ]`

Parameter	Value	Default
toggle	`"string"`	`undefined`
lambda_scaling	`real`	`undefined`

Summary Applies a contact heat flux of the form $\lambda\sigma\epsilon(T_1^4 - T_2^4)/2$ to sides 1 and 2 of the contact interface where λ is a scaling parameter, σ is the stefan boltzmann constant ϵ is the emissivity. This model is similar to the DASH_TIED model but for radiation rather than convection.

24.4.23 Enforcement For Energy = Tied D Style Temperature

Scope: Enforcement

`Enforcement For Energy = Tied D Style Temperature [ Toggle =` *`toggle`* `|Lambda_Scaling = lambda_scaling ]`

Parameter	Value	Default
toggle	`"string"`	`undefined`
lambda_scaling	`real`	`undefined`

Summary Applies a contact heat flux of the form $\lambda\bar{h}(T_1 - T_2)/2$ to sides 1 and 2 of the contact interface where λ is a penalty parameter, $\bar{h}$ is an average conductance coefficient. This model is similar to the DASH_TIED model.

24.4.24 Enforcement For Energy = Tied Ip Temperature

Scope: Enforcement

`Enforcement For Energy = Tied Ip Temperature [ Toggle = toggle | Co_Field = co_field ]`

Parameter	Value	Default
toggle	"string"	undefined
co_field	integer	undefined

Summary Applies a contact heat flux of the form $(k_1 \nabla T_1 \cdot n_1 + k_2 \nabla T_2 \cdot n_2 + \bar{h}(T_1 - T_2))/2$ to sides 1 and 2 of the contact interface where k is the thermal conductivity and $\bar{h}$ is an average conductance coefficient. Additionally an interior penalty jump term is also applied.

24.4.25 Enforcement For Energy = Tied Temperature

Scope: Enforcement

`Enforcement For Energy = Tied Temperature [ Toggle = toggle | Co_Field = co_field ]`

Parameter	Value	Default
toggle	"string"	undefined
co_field	integer	undefined

Summary Applies a contact heat flux of the form $(k_1 \nabla T_1 \cdot n_1 + k_2 \nabla T_2 \cdot n_2 + \bar{h}(T_1 - T_2))/2$ to sides 1 and 2 of the contact interface where k is the thermal conductivity and $\bar{h}$ is an average conductance coefficient.

24.4.26 Enforcement For Hfem Continuity = Tied Ip Continuity

Scope: Enforcement

24.4.27 Enforcement For Hfem Momentum = Tied Ip Momentum

Scope: Enforcement

24.4.28 Enforcement For Hfem Momentum = Tied Momentum

Scope: Enforcement

24.4.29 Enforcement For Lubrication Height = Deforming

Scope: Enforcement

`Enforcement For Lubrication Height = Deforming [ Hmin = hmin ]`

Parameter	Value	Default
hmin	real	undefined

Summary Lubrication height that deforms by the displacement of the adjoining volume region.

24.4.30 Enforcement For Mass Balance = Bulk Node Mass Open

Scope: Enforcement

`Enforcement For Mass Balance = Bulk Node Mass Open [ Bulk_Node = bulk_node ]`

Parameter	Value	Default
bulk_node	"string"	undefined

Summary Open BC between a porous medium and a bulk node

24.4.31 Enforcement For Mass Balance = Bulk Node Open

Scope: Enforcement

`Enforcement For Mass Balance = Bulk Node Open [ Bulk_Node = bulk_node ]`

Parameter	Value	Default
bulk_node	"string"	undefined

24.4.32 Enforcement For Mass Balance = Bulk Node Species Open

Scope: Enforcement

`Enforcement For Mass Balance = Bulk Node Species Open [ Bulk_Node = bulk_node ]`

Parameter	Value	Default
bulk_node	"string"	undefined

24.4.33 Enforcement For Mass Balance = Conserved Mass

Scope: Enforcement

`Enforcement For Mass Balance = Conserved Mass [ Co_Field = co_field ]`

Parameter	Value	Default
co_field	integer	undefined

Summary Continuous mass balance and capillary pressure, discontinuous nonwetting phase saturation.

24.4.34 Enforcement For Mesh = Lubrication Pressure

Scope: Enforcement

Summary Lubrication pressure mesh flux

24.4.35 Enforcement For Momentum = Tied Ip Momentum

Scope: Enforcement

24.4.36 Enforcement For Momentum = Tied Momentum

Scope: Enforcement

24.4.37 Enforcement For Porous Enthalpy = Bulk Node Open

Scope: Enforcement

`Enforcement For Porous Enthalpy = Bulk Node Open [ Bulk_Node =` *`bulk_node`* `]`

Parameter	Value	Default
bulk_node	"string"	undefined

24.4.38 Enforcement For Porous Enthalpy = Bulk Node Open Uncoupled

Scope: Enforcement

`Enforcement For Porous Enthalpy = Bulk Node Open Uncoupled [ Bulk_Node =` *`bulk_node`* `]`

Parameter	Value	Default
bulk_node	"string"	undefined

24.4.39 Enforcement For Porous Enthalpy = Contact Resistance

Scope: Enforcement

`Enforcement For Porous Enthalpy = Contact Resistance [ Opposite_Material_Phase =` *`opposite_material_ph`*
`| Bulk_Node =` *`bulk_node`* `]`

Parameter	Value	Default
opposite_material_phase	"string"	undefined
bulk_node	"string"	undefined

Summary Applies a heat flux of the form $J \cdot n = h(T - T_{opp})$ where h is the gap conductance coefficient defined for the surface material and the material phase of T_{opp} is optionally specified.

24.4.40 Enforcement For Porous Species = Bulk Node Open All Inflow

Scope: Enforcement

`Enforcement For Porous Species = Bulk Node Open All Inflow [ Bulk_Node =` *`bulk_node`* `]`

Parameter	Value	Default
bulk_node	"string"	undefined

24.4.41 Enforcement For Voltage = Tied Voltage

Scope: Enforcement

`Enforcement For Voltage = Tied Voltage [ Toggle =` *`toggle`* `| Co_Field =` *`co_field`* `]`

Parameter	Value	Default
toggle	"string"	undefined
co_field	integer	undefined

Summary Applies a contact electric displacement flux of the form $(\epsilon_1 \nabla V_1 \cdot n_1 + \epsilon_2 \nabla V_2 \cdot n_2 + \bar{C}(V_1 - V_2))/2$ to sides 1 and 2 of the contact interface where ϵ is the electrical permittivity and $\bar{C}$ is an average conductance coefficient.

24.5 Interaction

Scope: Contact Definition

```
Begin Interaction Interaction_name
    Normal Tolerance {= | are | is} Distance
    Surfaces {= | are | is} Surfaces...
End
```

Summary Contains the commands needed to define a surface-surface interaction.

24.5.1 Normal Tolerance

Scope: Interaction

```
Normal Tolerance {= | are | is} Distance
```

Parameter	Value	Default
Distance	real	undefined

Summary Set distance of normal tolerance.

24.5.2 Surfaces

Scope: Interaction

```
Surfaces {= | are | is} Surfaces...
```

Parameter	Value	Default
Surfaces	string...	undefined

Summary Defines a pairwise set of surfaces for which to turn on interactions

24.5.3 Normal Tolerance

Scope:

```
Normal Tolerance {= | are | is} Distance
```

Parameter	Value	Default
Distance	`real`	`undefined`

Summary Set distance of normal tolerance.

24.6 Search Options

Scope: Contact Definition

```
Begin Search Options Search_options_name
    Normal Tolerance {= | are | is} Distance
    Search Method {= | are | is} SearchMethod
End
```

Summary Contains the commands to set the search options

24.6.1 Normal Tolerance

Scope: Search Options

```
Normal Tolerance {= | are | is} Distance
```

Parameter	Value	Default
Distance	`real`	`undefined`

Summary Set distance of normal tolerance.

24.6.2 Search Method

Scope: Search Options

```
Search Method {= | are | is} SearchMethod
```

Parameter	Value	Default
SearchMethod	`{boost	kdtree}`	`NONE`

Summary Set the search method to be used with nonconformal contact.

24.6.3 Normal Tolerance

Scope:

```
Normal Tolerance {= | are | is} Distance
```

Parameter	Value	Default
Distance	`real`	`undefined`

Summary Set distance of normal tolerance.

Chapter 25

Output Reference

25.1 Output Overview

Output is divided into three major categories, Results output, Heartbeat output and Restart output. Results output contains binary format simulation results in a form suitable for visualization. While the Results output usually corresponds to the entire model, it can also be applied to portions of the model. Heartbeat output is generally written to a text file and provides a convenient means of monitoring intermediate simulation results. Restart output (Restart Data) is specialized for use in analyses that may not complete in a single job submission but might complete when continued from a previous termination time.

When using Results output or Heartbeat output one should note that the output request must be made using the internal Field name 2.10, most frequently solution->field_name or pp->postprocess_name. If the internal Field name is not used in the these requests output cannot be generated for that Field and a warning to that effect will be written to the log file. Global variables may also be output to the Results and Heartbeat output files. A list of all fields and global variables available for output in a particular model is output to the log file immediately after the mesh is read. This list can be previewed by running aria with the –check-input option that will load verify the input syntax, read the mesh, print the variables available for output, and then exit.

Depending upon the application one may wish to define multiple Results output or Heartbeat output command blocks. In doing so one must recognize that output time or output interval specifications are cumulative over the various output requests. Note that because time stepping accommodates simulation output requests, inconsistency of output frequency specification among multiple Results output blocks can produce unexpected/undesirable behavior in the solution time stepping.

Commands pertinent to the definition of Restart are included in this section. These commands describe management of restart files and content of the files. The intent of restart is to begin a new sequence of calculations using the entire content of the restart file. Those wishing to restart a simulation using only part of the results may wish to consider using the IC READ_FILE option 8.7. The invocation of restart within a simulation (specifying when it should start) is outlined elsewhere in this document 3.10.

Within the input file Output command blocks will appear at the Region scope as illustrated below.

```
.
.
.
Begin Procedure My_Aria_Procedure
   .
   .
   Begin Aria Region My_Region
    .
    .
    Begin Results output my_output
```

```
      .
      .
    End Results Output my_output

    Begin Heartbeat Output my_values
      .
      .
    End Heartbeat Output

    Begin Restart output go_again
      .
      .
    End Restart output
    .
    .
   End

End
.
.
```

25.2 Results Output

Scope: Encore Region

```
Begin Results Output Label
    Additional Steps {=|are|is} List_of_steps...
    Additional Times {=|are|is} List_of_times...
    At Step n Option {=|are|is} m
    At Time Dt1 Option {=|are|is} Dt2
    Component Separator Character {=|are|is} Separator
    Database Name {=|are|is} StreamName
    Database Type {=|are|is} DatabaseType
    Edge [ VariableList... ]
    Edge Variables {=|are|is} [ VariableList... ]
    Element [ VariableList... ]
    Element Variables {=|are|is} [ VariableList... ]
    Enable Large Ids
    Exclude {=|are|is} [ ElementBlockList... ]
    Exists Option1 Option2
    Face [ VariableList... ]
    Face Variables {=|are|is} [ VariableList... ]
    Global [ Variables... ]
    Global Variables {=|are|is} [ Variables... ]
    Include {=|are|is} [ ElementBlockList... ]
    Nodal [ VariableList... ]
```

```
    Nodal Variables {=|are|is} [ VariableList...  ]
    Node [ VariableList...  ]
    Node Variables {=|are|is} [ VariableList...  ]
    Nodeset [ VariableList...  ]
    Nodeset Variables {=|are|is} [ VariableList...  ]
    Output Mesh {=|are|is} OutputMesh
    Output On Signal {=|are|is} Signals
    Overwrite Option1 Option2
    Property PropertyName {=|are|is} PropertyValue
    Sideset [ VariableList...  ]
    Sideset Variables {=|are|is} [ VariableList...  ]
    Start Time {=|are|is} Start_time
    Surface [ VariableList...  ]
    Surface Variables {=|are|is} [ VariableList...  ]
    Synchronize Output
    Termination Time {=|are|is} Final_time
    Timeseries Name {=|are|is} filename
    Timestep Adjustment Interval {=|are|is} Nsteps
    Title
    Use Output Scheduler Timer_name
    Begin Catalyst Label
    End

End
```

Summary Describes the location and type of the output stream used for outputting results for the enclosing region.

25.2.1 Additional Steps

Scope: Results Output

```
Additional Steps {=|are|is} List_of_steps...
```

Parameter	Value	Default
List_of_steps	integer...	undefined

Summary Additional simulation steps when output should occur.

25.2.2 Additional Times

Scope: Results Output

`Additional Times {=|are|is} List_of_times...`

Parameter	Value	Default
List_of_times	`real...`	`undefined`

Summary Additional simulation times when output should occur.

25.2.3 At Step

Scope: Results Output

`At Step n Option {=|are|is} m`

Parameter	Value	Default
n	`integer`	`undefined`
m	`integer`	`undefined`

Summary Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

25.2.4 At Time

Scope: Results Output

`At Time Dt1 Option {=|are|is} Dt2`

Parameter	Value	Default
Dt1	`real`	`undefined`
Dt2	`real`	`undefined`

Summary Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

25.2.5 Component Separator Character

Scope: Results Output

`Component Separator Character {=|are|is} Separator`

Parameter	Value	Default
Separator	`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.

25.2.6 Database Name

Scope: Results Output

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

Parameter	Value	Default
StreamName	`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 ".e" suffix appended.

25.2.7 Database Type

Scope: Results Output

Summary The database type/format to be used for the output results.

25.2.8 Edge

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Edge variables are not supported for all database types.

25.2.9 Edge Variables

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Edge variables are not supported for all database types.

25.2.10 Element

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities"

25.2.11 Element Variables

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities"

25.2.12 Enable Large Ids

Scope: Results Output

Summary Enable 64 bit entity IDs for output

25.2.13 Exclude

Scope: Results Output

Summary Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will not be output to the results database.

25.2.14 Exists

Scope: Results Output

Summary Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is "OVERWRITE" which deletes the existing file and creates a new file of the same name. "APPEND" will (if possible) append the new data to the end of the existing file. "ABORT" will print an error message and end the analysis. "ADD_SUFFIX" will add a -s???? suffix where the ???? is replaced by a sequential number starting at 0002.

25.2.15 Face

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Face variables are not supported for all database types.

25.2.16 Face Variables

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Face variables are not supported for all database types.

25.2.17 Global

Scope: Results Output

Summary Define the global variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line.

25.2.18 Global Variables

Scope: Results Output

Summary Define the global variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line.

25.2.19 Include

Scope: Results Output

Summary Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will be output to the results database.

25.2.20 Nodal

Scope: Results Output

Summary Define the nodal variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line.

25.2.21 Nodal Variables

Scope: Results Output

Summary Define the nodal variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line.

25.2.22 Node

Scope: Results Output

Summary Define the nodal variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line.

25.2.23 Node Variables

Scope: Results Output

Summary Define the nodal variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line.

25.2.24 Nodeset

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Nodeset variables are not supported for all database types.

25.2.25 Nodeset Variables

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Nodeset variables are not supported for all database types.

25.2.26 Output Mesh

Scope: Results Output

`Output Mesh {=|are|is}` *`OutputMesh`*

Parameter	Value	Default
OutputMesh	`{exposed surface|refined|unrefined}`	`undefined`

Summary Use this command to turn on "unrefined" as the output mesh. The default behavior is "refined", in which field variables are output on the current mesh, which may have been refined (either uniformly or adaptively) or had its topology altered in some way (e.g., dynamic load balancing) with respect to the original mesh read from the the input file. By specifying "Output Mesh = unrefined", all output variables are output only on the original mesh objects read from the input file.

25.2.27 Output On Signal

Scope: Results Output

`Output On Signal {=|are|is}` *`Signals`*

Parameter	Value	Default
Signals	`{sigabrt|sigalrm|sigfpe|sighup|sigill|sigint|sigkill|sigpipe|sigquit|sigsegv|sigterm|sigusr1|sigusr2}`	`undefined`

Summary When the specified signal is raised, the output stream associated with this block will be output.

25.2.28 Overwrite

Scope: Results Output

Summary (DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

25.2.29 Property

Scope: Results Output

`Property` *`PropertyName`* `{=|are|is}` *`PropertyValue`*

Parameter	Value	Default
PropertyName	`string`	`undefined`
PropertyValue	`string`	`undefined`

Summary Define a database property named "PropertyName" with the value "PropertyValue". If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is "true" or "yes" or "false" or "no", it will define a logical property; otherwise it will define a string property. Suppported properties are typically database dependent; Current properties are: COMPRESSION_LEVEL = [0..9] (off) COMPRESSION_SHUFFLE = true|false|on|off (off) FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file) (netcdf3) INTEGER_SIZE_DB = 4|8 (4) INTEGER_SIZE_API = 4|8 (4) REAL_SIZE_DB = 4|8 (8 is default) LOGGING = true|false|on|off (off) MAX_NAME_LENGTH = value (32)

25.2.30 Sideset

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Face variables are not supported for all database types.

25.2.31 Sideset Variables

Scope: Results Output

Summary Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Face variables are not supported for all database types.

25.2.32 Start Time

Scope: Results Output

`Start Time {= | are | is}` *Start_time*

Parameter	Value	Default
Start_time	`real`	`undefined`

Summary

Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

25.2.33 Surface

Scope: Results Output

Summary

Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Face variables are not supported for all database types.

25.2.34 Surface Variables

Scope: Results Output

Summary

Define the variables that should be written to the results database. If "variable" is entered, then its name will be used on the output database. If "variable as db_name" is entered, then "db_name" will be the name used on the database for the internal variable "variable". Multiple "variable" or "variable as db_name" entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with "exclude list_of_entities" or "include list_of_entities". Face variables are not supported for all database types.

25.2.35 Synchronize Output

Scope: Results Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

25.2.36 Termination Time

Scope: Results Output

`Termination Time {= | are | is}` *`Final_time`*

Parameter	Value	Default
Final_time	`real`	`undefined`

Summary Specify the time to stop outputting results from this output request block.

25.2.37 Timeseries Name

Scope: Results Output

`Timeseries Name {= | are | is}` *`filename`*

Parameter	Value	Default
filename	`string`	`undefined`

Summary Optionally specify a filename for a timeseries file that outputs the root database filename in the order that they are written. This is useful when running on large numbers of processors with many mesh-mods that cause simple disk operations to hang.

25.2.38 Timestep Adjustment Interval

Scope: Results Output

`Timestep Adjustment Interval {= | are | is}` *`Nsteps`*

Parameter	Value	Default
Nsteps	`integer`	`undefined`

Summary Specify the number of steps to 'look ahead' and adjust the timestep to ensure that the specified output times or simulation end time will be hit 'exactly'.

25.2.39 Title

Scope: Results Output

Summary Specify the title to be used for this specific output block.

25.2.40 Use Output Scheduler

Scope: Results Output

```
Use Output Scheduler Timer_name
```

Parameter	Value	Default
Timer_name	string	undefined

Summary Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

25.3 Heartbeat

Scope: Encore Region

```
Begin Heartbeat Label
    Additional Steps {=|are|is} List_of_steps...
    Additional Times {=|are|is} List_of_times...
    Append {=|are|is} Option
    At Step n Option {=|are|is} m
    At Time Dt1 Option {=|are|is} Dt2
    Element [ VariableList... ]
    Exists Option1 Option2
    Face [ VariableList... ]
    Flush Interval {=|are|is} Option
    Format {=|are|is} StreamTypes
    Global [ Variables... ]
    Labels {=|are|is} Option
    Legend {=|are|is} Option
    Monitor Equals Option
    Nodal [ VariableList... ]
    Node [ VariableList... ]
    Nodeset [ VariableList... ]
    Output On Signal {=|are|is} Signals
    Precision {=|are|is} Precision
    Start Time {=|are|is} Start_time
    Stream Name {=|are|is} OutputFilename
    Synchronize Output
    Termination Time {=|are|is} Final_time
    Timestamp Format
    Timestep Adjustment Interval {=|are|is} Nsteps
    Use Output Scheduler Timer_name
    Variable {=|are|is} Option Variable_list...
End
```

Summary Describes the location and type of the output stream used for outputting the heartbeat information for the enclosing region.

25.3.1 Additional Steps

Scope: Heartbeat

`Additional Steps {=|are|is}` *`List_of_steps...`*

Parameter	Value	Default
List_of_steps	`integer...`	`undefined`

Summary Additional simulation steps when output should occur.

25.3.2 Additional Times

Scope: Heartbeat

`Additional Times {=|are|is}` *`List_of_times...`*

Parameter	Value	Default
List_of_times	`real...`	`undefined`

Summary Additional simulation times when output should occur.

25.3.3 Append

Scope: Heartbeat

Summary Specifies whether the heartbeat file is appended if it exists. By default, the file is appended if restart is requested and not if restart is not requested. This option does not work for automatic restarts because a new heartbeat file is written with each auto restart.

25.3.4 At Step

Scope: Heartbeat

`At Step` *`n Option`* `{=|are|is}` *`m`*

Parameter	Value	Default
n	`integer`	`undefined`
m	`integer`	`undefined`

Summary Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

25.3.5 At Time

Scope: Heartbeat

`At Time` *`Dt1 Option`* `{=|are|is}` *`Dt2`*

Parameter	Value	Default
Dt1	`real`	`undefined`
Dt2	`real`	`undefined`

Summary Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

25.3.6 Element

Scope: Heartbeat

Summary Define the element variables that should be written to the heartbeat database. The syntax is: "element internal_name at element id as DBname" or "element internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the heartbeat database.

25.3.7 Exists

Scope: Heartbeat

Summary Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is "OVERWRITE" which deletes the existing file and creates a new file of the same name. "APPEND" will (if possible) append the new data to the end of the existing file. "ABORT" will print an error message and end the analysis.

25.3.8 Face

Scope: Heartbeat

Summary Define the face variables that should be written to the heartbeat database. The syntax is: "face internal_name at face id as DBname" or "face internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the heartbeat database.

25.3.9 Flush Interval

Scope: Heartbeat

`Flush Interval {=|are|is}` *`Option`*

Parameter	Value	Default
Option	`integer`	10

Summary The minimum time interval (in seconds) at which the heartbeat output will be explicitly flushed to disk. The default is 10 seconds.

25.3.10 Format

Scope: Heartbeat

`Format {= | are | is}` *StreamTypes*

Parameter	Value	Default
StreamTypes	`{csv	original	spyhis}`	`undefined`

Summary The stream type/format to be used for the output results.The only three options at this time are 'Original' which is the old default Sierra heartbeat format; 'SpyHis' which mimics the CTH Spyhis history output format; and 'CSV'

25.3.11 Global

Scope: Heartbeat

Summary Define the global/reduction variables that should be written to the heartbeat database. The syntax is: "global internal_name as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the heartbeat database.

25.3.12 Labels

Scope: Heartbeat

Summary Specifies whether labels will be displayed or just the value of the variable. Labels will be shown if this line is not present.

25.3.13 Legend

Scope: Heartbeat

Summary Specifies whether a legend will be displayed prior to outputting any variables. The legend will not be shown unless this line is present. The legend shows the names of the variables that will be written to the heartbeat output stream. If the variable has multiple components, then the component count is shown after the variable e.g., velocity(3).

25.3.14 Monitor

Scope: Heartbeat

Summary Specifies whether a line will be written to the heartbeat stream when either the results, history, and/or restart data are output.

25.3.15 Nodal

Scope: Heartbeat

Summary — Define the nodal variables that should be written to the heartbeat database. The syntax is: "nodal internal_name at node id as DBname" or "nodal internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the heartbeat database.

25.3.16 Node

Scope: Heartbeat

Summary — Define the nodal variables that should be written to the heartbeat database. The syntax is: "node internal_name at node id as DBname" or "node internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the heartbeat database.

25.3.17 Nodeset

Scope: Heartbeat

Summary — Define the nodeset variables that should be written to the heartbeat database. The syntax is: "nodeset internal_name at node id as DBname" or "nodeset internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the heartbeat database.

25.3.18 Output On Signal

Scope: Heartbeat

`Output On Signal {= | are | is}` *`Signals`*

Parameter	Value	Default
Signals	`{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}`	`undefined`

Summary — When the specified signal is raised, the output stream associated with this block will be output.

25.3.19 Precision

Scope: Heartbeat

`Precision {=|are|is} Precision`

Parameter	Value	Default
Precision	integer	5

Summary The precision to be used for the output of real variables (default=5).

25.3.20 Start Time

Scope: Heartbeat

`Start Time {=|are|is} Start_time`

Parameter	Value	Default
Start_time	real	undefined

Summary Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

25.3.21 Stream Name

Scope: Heartbeat

`Stream Name {=|are|is} OutputFilename`

Parameter	Value	Default
OutputFilename	string	undefined

Summary The filename of where the heartbeat data should be written. 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. In addition, there are several predefined streams that can be specified. The predefined streams are 'cout' or 'stdout' specifies standard output; 'cerr', 'stderr', 'clog', or 'log' specifies standard error; 'output' or 'outputP0' specifies Sierra's standard output which is redirected to the file specified by the '-o' option on the command line. If the file already exists, it is overwritten. If this line is omitted, then a filename will be created from the basename of the input file with a ".hrt" suffix appended.

25.3.22 Synchronize Output

Scope: Heartbeat

Summary In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are

not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

25.3.23 Termination Time

Scope: Heartbeat

`Termination Time {=|are|is}` *`Final_time`*

Parameter	Value	Default
Final_time	`real`	`undefined`

Summary Specify the time to stop outputting results from this output request block.

25.3.24 Timestamp Format

Scope: Heartbeat

Summary The format to be used for the timestamp. See 'man strftime' for more information.

25.3.25 Timestep Adjustment Interval

Scope: Heartbeat

`Timestep Adjustment Interval {=|are|is}` *`Nsteps`*

Parameter	Value	Default
Nsteps	`integer`	`undefined`

Summary Specify the number of steps to 'look ahead' and adjust the timestep to ensure that the specified output times or simulation end time will be hit 'exactly'.

25.3.26 Use Output Scheduler

Scope: Heartbeat

`Use Output Scheduler` *`Timer_name`*

Parameter	Value	Default
Timer_name	`string`	`undefined`

Summary Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

25.3.27 Variable

Scope: Heartbeat

`Variable {=|are|is} Option Variable_list...`

Parameter	Value	Default
Variable_list	string...	undefined

Summary

Define the variables that should be written to the heartbeat output. The user can request that the values of certain variables be output on the heartbeat line. These variables are limited to region and framework control data currently. The syntax is:

```
variable = {entity_type} {internal_name} at
           {entity_type} {entity_id}     as {external_name}
variable = {entity_type} {internal_name} nearest location
           {x,y,z} as {external_name}
```

For global variables, use:

```
variable = global {internal_name} [as {external_name}]
```

Where:

```
entity_type = node, element, face, edge, global
internal_name = Sierra variable name
entity_id = id of the node, element, face, edge that you want
    the specified variable output at.
external_name = name of variable on the database.
```

The names 'timestep', and 'time' can be specified as variables also. They are the current timestep and simulation time. This line can appear multiple times.

25.4 History Output

Scope: Encore Region

`Begin History Output Label`

`Additional Steps {=|are|is} List_of_steps...`

`Additional Times {=|are|is} List_of_times...`

`At Step n Option {=|are|is} m`

`At Time Dt1 Option {=|are|is} Dt2`

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

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

`Element [ VariableList... ]`

`Exists Option1 Option2`

`Face [ VariableList... ]`

`Global [ Variables... ]`

`Nodal [ VariableList... ]`

```
    Node [ VariableList...  ]
    Nodeset [ VariableList...  ]
    Output On Signal {=|are|is} Signals
    Overwrite Option1 Option2
    Property PropertyName {=|are|is} PropertyValue
    Start Time {=|are|is} Start_time
    Synchronize Output
    Termination Time {=|are|is} Final_time
    Timestep Adjustment Interval {=|are|is} Nsteps
    Title
    Use Output Scheduler Timer_name
    Variable {=|are|is} Option Variable_list...
End
```

Summary Describes the location and type of the output stream used for outputting history for the enclosing region.

25.4.1 Additional Steps

Scope: History Output

```
Additional Steps {=|are|is} List_of_steps...
```

Parameter	Value	Default
List_of_steps	integer...	undefined

Summary Additional simulation steps when output should occur.

25.4.2 Additional Times

Scope: History Output

```
Additional Times {=|are|is} List_of_times...
```

Parameter	Value	Default
List_of_times	real...	undefined

Summary Additional simulation times when output should occur.

25.4.3 At Step

Scope: History Output

```
At Step n Option {=|are|is} m
```

Parameter	Value	Default
n	integer	undefined
m	integer	undefined

Summary Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

25.4.4 At Time

Scope: History Output

`At Time` *`Dt1 Option`* `{= | are | is}` *`Dt2`*

Parameter	Value	Default
Dt1	real	undefined
Dt2	real	undefined

Summary Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

25.4.5 Database Name

Scope: History Output

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

Parameter	Value	Default
StreamName	string	undefined

Summary The base name of the database containing the output history. 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 ".h" suffix appended.

25.4.6 Database Type

Scope: History Output

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

Parameter	Value	Default
DatabaseTypes	{catalyst | dof | dof_exodus | exodus | exodusii | generated | genesis | parallel_exodus}	undefined

Summary The database type/format to be used for the output history.

25.4.7 Element

Scope: History Output

Summary Define the element variables that should be written to the history database. The syntax is: "element internal_name at element id as DBname" or "element internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the history database.

25.4.8 Exists

Scope: History Output

Summary Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is "OVERWRITE" which deletes the existing file and creates a new file of the same name. "APPEND" will (if possible) append the new data to the end of the existing file. "ABORT" will print an error message and end the analysis. "ADD_SUFFIX" will add a -s???? suffix where the ???? is replaced by a sequential number starting at 0002.

25.4.9 Face

Scope: History Output

Summary Define the face variables that should be written to the history database. The syntax is: "face internal_name at face id as DBname" or "face internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the history database.

25.4.10 Global

Scope: History Output

Summary Define the global/reduction variables that should be written to the history database. The syntax is: "global internal_name as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the history database.

25.4.11 Nodal

Scope: History Output

Summary Define the nodal variables that should be written to the history database. The syntax is: "nodal internal_name at node id as DBname" or "nodal internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the history database.

25.4.12 Node

Scope: History Output

Summary — Define the nodal variables that should be written to the history database. The syntax is: "node internal_name at node id as DBname" or "node internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the history database.

25.4.13 Nodeset

Scope: History Output

Summary — Define the nodeset variables that should be written to the history database. The syntax is: "nodeset internal_name at node id as DBname" or "nodeset internal_name nearest location X, Y, Z as DBname".

Where internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the history database.

25.4.14 Output On Signal

Scope: History Output

`Output On Signal {= | are | is}` *`Signals`*

Parameter	Value	Default
Signals	{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}	undefined

---

Summary — When the specified signal is raised, the output stream associated with this block will be output.

25.4.15 Overwrite

Scope: History Output

Summary — (DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

25.4.16 Property

Scope: History Output

`Property` *`PropertyName`* `{= | are | is}` *`PropertyValue`*

Parameter	Value	Default
PropertyName	`string`	`undefined`
PropertyValue	`string`	`undefined`

Summary Define a database property named "PropertyName" with the value "PropertyValue". If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is "true" or "yes" or "false" or "no", it will define a logical property; otherwise it will define a string property. Suppported properties are typically database dependent; Some history-related properties are: VARIABLE_NAME_CASE = upper|lower MAX_NAME_LENGTH = value (32)

25.4.17 Start Time

Scope: History Output

`Start Time {=|are|is}` *`Start_time`*

Parameter	Value	Default
Start_time	`real`	`undefined`

Summary Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

25.4.18 Synchronize Output

Scope: History Output

Summary In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

25.4.19 Termination Time

Scope: History Output

`Termination Time {=|are|is}` *`Final_time`*

Parameter	Value	Default
Final_time	`real`	`undefined`

Summary Specify the time to stop outputting results from this output request block.

25.4.20 Timestep Adjustment Interval

Scope: History Output

`Timestep Adjustment Interval {=|are|is}` *`Nsteps`*

Parameter	Value	Default
Nsteps	integer	undefined

Summary Specify the number of steps to 'look ahead' and adjust the timestep to ensure that the specified output times or simulation end time will be hit 'exactly'.

25.4.21 Title

Scope: History Output

Summary Specify the title to be used for this specific output block.

25.4.22 Use Output Scheduler

Scope: History Output

`Use Output Scheduler` *`Timer_name`*

Parameter	Value	Default
Timer_name	string	undefined

Summary Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

25.4.23 Variable

Scope: History Output

`Variable {=|are|is}` *`Option Variable_list...`*

Parameter	Value	Default
Variable_list	string...	undefined

Summary Define the variables that should be written to the history database. The syntax is: "variable = entity internal_name at entity id as DBname" or "variable = entity internal_name nearest location X, Y, Z as DBname" or "variable = entity internal_name at location X, Y, Z as DBname".

Where entity is 'node', 'element', 'face', or 'edge'; internal_name is the name of the variable in the Sierra application; and DBname is the name as it should appear on the history database.

25.5 Data Probe

Scope: Equation System

```
Begin Data Probe probeName

    Nodal_Id fieldName Node_Id {=|are|is} node_id [ Label label ]

    Nodal_Location fieldName Location {=|are|is} location1 location2[ location3] [ Label
    label ]

    Parametric Tolerance parametric_tolerance...

    Restriction Parts restriction_parts...

    Search Method {=|are|is} SearchMethod

    At Step startingStep Increment {=|are|is} increment

    Output To File [ fileName ]

End
```

Summary The Aria Data Probe capability allows the analyst to query data for fields of interest (e.g. temperature, pressure) during a simulation at times and locations specified by the analyst.

Within the Data Probe a user can request multiple probes but each Data Probe command block is restricted to processing of a single field.

Each individual probe within the Data Probe can be assigned a unique label by which can be used to access the probed value as a global variable. If a probe label is not defined then the probe will be internally labeled as the Data Probe name appended with an counter index (_index) that reflects the order in which the unnamed probe was parsed.

This command creates an Aria Data Probe with the name given by probeName and each probe point can be accessed and output as a global variable.

In problems of mesh motion only Nodal_Id probe points are supported.

25.5.1 Nodal_Id

Scope: Data Probe

```
Nodal_Id fieldName Node_Id {=|are|is} node_id [ Label label ]
```

Parameter	Value	Default
fieldName	string	undefined
node_id	integer	undefined

Summary Defines data probe for a point specified by the global id of a node.

Description Sets the nodal field name, node global id, and optionally the label for each probe point.

Probe requests of solution DOF will require the appropriate prefix, *solution–>* or *nonlinear_solution–>*.

The field_data at the specified node (and only that node) is associated with the probe point and its corresponding global variable.

If a label is provided, a global variable with that name will be created. If no label is provided, a global variable will be created with a name that is the concatenation of the Data Probe name and the index of the probe point.

25.5.2 Nodal_Location

Scope: Data Probe

Nodal_Location *fieldName* Location {=|are|is} $location_1$ $location_2$[$location_3$] [Label *label*]

Parameter	Value	Default
fieldName	string	undefined
location	real_1 real_2[real_3]	undefined

Summary

Defines data probe for a point specified by location (x,y,z).

Description

Sets the nodal field name, spatial location, and optionally the label for this probe point.

Probe requests of solution DOF will require the appropriate prefix, *solution–>* or *nonlinear_solution–>*.

If the point lies within an element within the set of element blocks blocks defined using RESTRICTION PARTS, the given field will be interpolated in that element. If the point lies outside the mesh, but within the PARAMETRIC TOLERANCE then the point is projected to the nearest location on the nearest element (face, edge, or vertex), and interpolation is used to compute the value of the field.

If a label is provided, a global variable with that name will be created. If no label is provided, a global variable will be created with a name that is the concatenation of the Data Probe name and the index of the probe point.

The NODAL LOCATION probe is not supported in problems with mesh motion.

25.5.3 Parametric Tolerance

Scope: Data Probe

Parametric Tolerance *parametric_tolerance...*

Parameter	Value	Default
parametric_tolerance	real...	undefined

Summary

The user may specify a parametric tolerance that is used to determine whether or not the data probe point is close enough to an element for the probe point to be considered valid. The parametric tolerance is a numeric value that indicates the maximum allowed parametric coordinate of the data probe point with respect to the closest element found during the search. The closest element of interest is transposed into isoparametric coordinates in the range of [-1,1] and the parametric distance is set as the maximum value. For an high aspect ratio hexahedron element, the parametric distance may look suspect as it will be calculated based on the shortest element edge. For instance, if the parametric tolerance is set to 2.0, the data probe point is allowed to have a parametric coordinate of up to 2.0 with respect to the closest element that is found. However, if the parametric coordinate of the point with respect to this element is > 2.0, the tolerance check will fail and the simulation will end with an error describing this failure. For convenience, all points within a probe are checked for the parametric tolerance condition, and if any failures are found, they are all reported together rather than the simulation halting on the first failure of this condition.

25.5.4 Restriction Parts

Scope: Data Probe

`Restriction Parts` *restriction_parts...*

Parameter	Value	Default
restriction_parts	`string...`	`undefined`

Summary Element blocks for which data probe will be applied. If the restriction part is a surface all element blocks touching the surface will be searched. Only elements within these parts will be searched for data probe points. If no restriction parts are provided, all element blocks will be searched.

25.5.5 Search Method

Scope: Data Probe

`Search Method {= | are | is}` *SearchMethod*

Parameter	Value	Default
SearchMethod	`{boost	kdtree}`	`BOOST_RTREE`

Summary Optionally select the search method used for finding the element containing the data probe point. Defaults to BOOST_RTREE.

25.5.6 At Step

Scope: Data Probe

`At Step` *startingStep* `Increment {= | are | is}` *increment*

Parameter	Value	Default
startingStep	`integer`	`undefined`
increment	`integer`	`undefined`

Summary Write the results of the data probe beginning at step (startingStep) and output every specified number of steps (increment).

25.5.7 Output To File

Scope: Data Probe

Summary Enable data output to file specified by filename. Data Probe Point values are additionally sent to the Aria log file.

25.6 Restart Overview

Convenient utilities are provided 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 from a solution plane is invoked using the RESTART TIME command line.

Note that special considerations are warranted when initializing problems utilizing bulk volume elements 31.4 with restart files not containing the bulk volume elements.

25.7 Restart Data

Scope: Encore Region

```
Begin Restart Data Label
    Additional Steps {=|are|is} List_of_steps...
    Additional Times {=|are|is} List_of_times...
    At Step n Option {=|are|is} m
    At Time Dt1 Option {=|are|is} Dt2
    At Wall Time Dt1 Option {=|are|is} Dt2
    Component Separator Character Option Separator
    Cycle Count {=|are|is} Count
    Database Name {=|are|is} StreamName
    Database Type {=|are|is} DatabaseTypes
    Debug Dump
    Decomposition Method {=|are|is} Method
    Exists Option1 Option2
    File Cycle Count {=|are|is} Count
    Input Database Name {=|are|is} StreamName
    Optional
    Output Database Name {=|are|is} StreamName
    Output On Signal {=|are|is} Signals
    Overlay Count {=|are|is} Count
    Overwrite Option1 Option2
    Property PropertyName {=|are|is} PropertyValue
    Restart {=|are|is} RestartOption
    Restart Time {=|are|is} Time
    Start Time {=|are|is} Start_time
    Synchronize Output
    Termination Time {=|are|is} Final_time
    Timestep Adjustment Interval {=|are|is} Nsteps
    Use Output Scheduler Timer_name
End
```

Summary Describes the data required to output and input restart data for the enclosing region.

25.7.1 Additional Steps

Scope: Restart Data

`Additional Steps {=|are|is}` *`List_of_steps...`*

Parameter	Value	Default
List_of_steps	`integer...`	`undefined`

Summary Additional simulation steps when output should occur.

25.7.2 Additional Times

Scope: Restart Data

`Additional Times {=|are|is}` *`List_of_times...`*

Parameter	Value	Default
List_of_times	`real...`	`undefined`

Summary Additional simulation times when output should occur.

25.7.3 At Step

Scope: Restart Data

`At Step` *`n Option`* `{=|are|is}` *`m`*

Parameter	Value	Default
n	`integer`	`undefined`
m	`integer`	`undefined`

Summary Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

25.7.4 At Time

Scope: Restart Data

`At Time` *`Dt1 Option`* `{=|are|is}` *`Dt2`*

Parameter	Value	Default
Dt1	`real`	`undefined`
Dt2	`real`	`undefined`

Summary Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

25.7.5 At Wall Time

Scope: Restart Data

`At Wall Time` *`Dt1 Option`* `{=|are|is}` *`Dt2`*

Parameter	Value	Default
Dt1	string	undefined
Dt2	string	undefined

Summary — Write a restart file at a specific wall time since the start of the run. Time string format allows s, m, h, d for seconds, minutes, hours, days

25.7.6 Component Separator Character

Scope: Restart Data

`Component Separator Character` *`Option Separator`*

Parameter	Value	Default
Separator	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.

25.7.7 Cycle Count

Scope: Restart Data

`Cycle Count {=|are|is}` *`Count`*

Parameter	Value	Default
Count	integer	undefined

Summary — Specify the number of restart steps which will be written to the restart database before previously written steps are overwritten. For example, if the cycle count is 5 and restart is written every 0.1 seconds, the restart system will write 0.1, 0.2, 0.3, 0.4, 0.5 to the database. It will then overwrite the first step with data from time 0.6, the second with time 0.7. At time 0.8, the database would contain data at times 0.6, 0.7, 0.8, 0.4, 0.5. Note that time will not necessarily be monotonically increasing on a database that specifies the cycle count.

25.7.8 Database Name

Scope: Restart Data

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

Parameter	Value	Default
StreamName	string	undefined

Summary The database containing the input and/or output restart data. If this analysis is being restarted, restart data will be read from this file. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists (after being read if applicable). 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. See also the 'Input Database' and 'Output Database' commands.

25.7.9 Database Type

Scope: Restart Data

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

Parameter	Value	Default
DatabaseTypes	`{catalyst | dof | dof_exodus | exodus | exodusii | generated | genesis | parallel_exodus}`	`undefined`

Summary The database type/format used for the restart file.

25.7.10 Debug Dump

Scope: Restart Data

Summary Specify whether the the restart system will write the restart data immediately after reading the restart data if the run is restarting. The output data can be compared with the restart input data to determine whether they match.

25.7.11 Decomposition Method

Scope: Restart Data

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

25.7.12 Exists

Scope: Restart Data

Summary Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is "OVERWRITE" which deletes the existing file and creates a new file of the same name. "APPEND" will (if possible) append the new data to the end of the existing file. "ABORT" will print an error message and end the analysis. "ADD_SUFFIX" will add a suffix to the file name and output to that file.

25.7.13 File Cycle Count

Scope: Restart Data

`File Cycle Count {=|are|is} Count`

Parameter	Value	Default
Count	integer	undefined

Summary Each restart dump will be written to a separate file suffixed with A,B, ... The count specifies how many separate files are used before the cycle repeats. For example, if "FILE CYCLE COUNT = 3" is specified, the restart dumps would be written to file-A.rs, file-B.rs, file-C.rs, file-A.rs, ... The maximum value for the cycle count is 26.

25.7.14 Input Database Name

Scope: Restart Data

`Input Database Name {=|are|is} StreamName`

Parameter	Value	Default
StreamName	string	undefined

Summary The database containing the input restart data. If this analysis is being restarted, restart data will be read from this file. See also the 'Database' and 'Output Database' commands.

25.7.15 Optional

Scope: Restart Data

Summary The database will be read if it exists, but it is not an error if there is no restart database to read for this region during a restarted analysis.

25.7.16 Output Database Name

Scope: Restart Data

`Output Database Name {=|are|is} StreamName`

Parameter	Value	Default
StreamName	string	undefined

Summary The database containing the output restart data. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists. See also the 'Database' and 'Input Database' commands.

25.7.17 Output On Signal

Scope: Restart Data

`Output On Signal {=|are|is} Signals`

Parameter	Value	Default
Signals	{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}	undefined

Summary When the specified signal is raised, the output stream associated with this block will be output.

25.7.18 Overlay Count

Scope: Restart Data

```
Overlay Count {= | are | is} Count
```

Parameter	Value	Default
Count	integer	undefined

Summary Specify the number of restart outputs which will be overlayed on top of the last written step. For example, if restarts are being output every 0.1 seconds and the overlay count is specified as 2, then restart will write times 0.1 to step 1 of the database. It will then write 0.2 and 0.3 also to step 1. It will then increment the database step and write 0.4 to step 2; overlay 0.5 and 0.6 on step 2... At the end of the analysis, assuming it runs to completion, the database would have times 0.3, 0.6, 0.9, ... However, if there were a problem during the analysis, the last step on the database would contain an intermediate step.

25.7.19 Overwrite

Scope: Restart Data

Summary (DEPRECATED, Use EXISTS) Specify whether the restart database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the restart block and either off, false, or no is specified.

25.7.20 Property

Scope: Restart Data

```
Property PropertyName {= | are | is} PropertyValue
```

Parameter	Value	Default
PropertyName	string	undefined
PropertyValue	string	undefined

Summary Define a database property named "PropertyName" with the value "PropertyValue". If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is "true" or "yes" or "false" or "no", it will define a logical property; otherwise it will define a string property. If PropertyName consists of multiple strings, they will be concated together with "_" separating the individual words. Suppported properties are typically database dependent; Current properties are: COMPRESSION_LEVEL = [0..9] COMPRESSION_SHUFFLE = true|false|on|off FILE_TYPE = netcdf4 (forces use of netcdf-4

hdf5-based file) INTEGER_SIZE_DB = 4|8 INTEGER_SIZE_API = 4|8 LOGGING = true|false|on|off MAX_NAME_LENGTH = value

25.7.21 Restart

Scope: Restart Data

Summary — Specify automatic restart file read.

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

Specify that the analysis should be restarted from the last common time on all restart databases for each Region in the analysis. 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.

By default, use of this command will not cause output files (e.g., results, history, heartbeat, restart) to be overwritten. Instead output files will be written with the same basename and the suffix `-s000*`. Common visualization packages are written to handle this file organization gracefully in order for the user to view all results seamlessly.

25.7.22 Restart Time

Scope: Restart Data

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

Parameter	Value	Default
Time	`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.

25.7.23 Start Time

Scope: Restart Data

`Start Time {= | are | is}` *`Start_time`*

Parameter	Value	Default
Start_time	`real`	`undefined`

Summary — Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

25.7.24 Synchronize Output

Scope: Restart Data

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

25.7.25 Termination Time

Scope: Restart Data

`Termination Time {=|are|is}` *`Final_time`*

Parameter	Value	Default
Final_time	`real`	`undefined`

Summary

Specify the time to stop outputting results from this output request block.

25.7.26 Timestep Adjustment Interval

Scope: Restart Data

`Timestep Adjustment Interval {=|are|is}` *`Nsteps`*

Parameter	Value	Default
Nsteps	`integer`	`undefined`

Summary

Specify the number of steps to 'look ahead' and adjust the timestep to ensure that the specified output times or simulation end time will be hit 'exactly'.

25.7.27 Use Output Scheduler

Scope: Restart Data

`Use Output Scheduler` *`Timer_name`*

Parameter	Value	Default
Timer_name	`string`	`undefined`

Summary Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

25.7.28 Restart Time

Scope:

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

Parameter	Value	Default
Time	`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.

Chapter 26

Input Output Region Reference

26.1 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 ability to input datasets that include the output of other simulations is provided. 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 15.1 and solution control 14. 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
```

26.2 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

26.2.1 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
Field_name	string	undefined
Dimension	integer	undefined

Summary Creates a Element Field name field_name on the region.

26.2.2 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
Field_name	string	undefined
Dimension	integer	undefined

Summary Creates a Nodal Field name field_name on the region.

26.2.3 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

26.2.4 Offset Time

Scope: Input_Output Region

Offset Time {=|are|is} *Period_offset_time*

Parameter	Value	Default
Period_offset_time	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

26.2.5 Periodicity Time

Scope: Input_Output Region

`Periodicity Time {=|are|is}` *`Periodicity_time`*

Parameter	Value	Default
Periodicity_time	`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

26.2.6 Start Time

Scope: Input_Output Region

`Start Time {=|are|is}` *`Start_time`*

Parameter	Value	Default
Start_time	`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

26.2.7 Use Finite Element Model

Scope: Input_Output Region

`Use Finite Element Model` *`ModelName`* `[ Model Coordinates Are` *`Nodal_variable_name`* `]`

Parameter	Value	Default
ModelName	`string`	`undefined`

Summary Associates a predefined finite element model with this region.

Chapter 27

Log File Output

27.1 Introduction

Aria, as well as all other Sierra codes, provides log file output capability which aids users in performing such actions as viewing run information, runtime/performance monitoring, and verifying input file accuracy. While much of the log file output is standardized for the Sierra codes, there are sections, especially in runtime monitoring, that are specific to Aria. This chapter will discuss the different sections of the log file and give examples of these sections in order to help the user assimilate and make the best use of the information that is output.

For this explanation, the Aria Foam Decomposition example problem will be used and can be found on the Sandia Computational Simulation webpage. This example was chosen due to its use of several different types of physics, such as element death, enclosure radiation, and chemistry modeling, as well as its use of adaptive time stepping.

27.2 Preamble

The log file preamble contains useful information about run information, input file parsing, and model setup.

27.2.1 Run Information

At the very beginning of the log file is printed information about the selected Sierra executable, where and when the model is run, and specific version information. Below is a section of the log file preamble which shows this information.

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

                                    Aria

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

                  Version 4.33.1-398-gaeb9bfbb

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

```
                    Sandia National Laboratories
                Albuquerque, New Mexico and Livermore, California

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


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

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

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

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

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

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

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

More specific version information for each product that Aria uses is shown at the bottom of this section, for those who are interested in or concerned with more detail about how their model is being run.

27.2.2 Input File

The log file next contains a complete copy of the input file which is used to run the model, directly following the lines shown below:

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
Reading foamDecomposition.i                                  May 14 2014 11:04:26
```

Note that the input file shown is the one that is used for parsing, so any pre-processing operations such as aprepro will have already been completed in the input file shown in the log file. Also, this section will be where parsing errors show up, if there are any in the input file.

In the event of misplacement of an input file, one can also extract the input file from the log file, which can save a considerable amount of time.

27.2.3 Instantiation

After the input file has been parsed, Aria will then instantiate all systems required to run the input file and will output the results of the instantiation to the log file. The instantiation begins with the lines

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
Instantiating system                                         May 14 2014 11:04:26
```

This section of the log file is not often used by analysts, but can be useful in debugging non-obvious issues. The following is a list of common items that may be seen in the instantiation section:

- Setting of default values for unspecified variables

  Example `DEFAULTING FAILED_TIME_STEP_SIZE_RATIO = 0.500000 for region AriaRegion in time block Transient_time_block_1`

- Configuration of mesh parts as needed by different inputs

  Example `Adding death bc iopart for element death with name surface_death_temp1`

- Creation of edge/element mechanics

  Example `Configuring Element mechanics for region "AriaRegion" equation system "main"...`

- Configuration of boundary/initial condition mechanics

Example `Configuring Convective Flux for region "AriaRegion" equation system "main"...`

- Configuration of other mechanics such as enclosure radiation and chemistry

  Example `Configuring Enclosure Radiation BC mechanics for region "AriaRegion" equation system "main"...`

The very end of this section of the log file contains the number of errors and warnings that were found during the parsing of the input file. This can be helpful in helping users pinpoint if they incorrectly set up some parameter that still allows the simulation to continue.

27.2.4 Procedure Initialization

This section of the log file begins with the following lines:

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
Running procedure AriaProcedure                                  May 14 2014 11:04:26
Reading mesh                                                     May 14 2014 11:04:26
```

The most useful portion of this section is the list of variables available for output. These variables are shown for each Region that is found in the input file and are separated by the type of field on which they reside, e.g., nodes, elements, edges, and faces. Global variables that may be output are also listed.

27.3 Run-time Reporting

After the preamble of the log file comes the actual execution of the model, including linear and nonlinear solver information, time stepping, and variable calculation. The information contained in this section can assist the user in determining whether the model is stable and if the correct results are being obtained. To illustrate this section of the log file, one time step of the run is shown below and will be further explained in the following sections.

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
Minimum Time Step Selection:
        min           max
Time step selection: dt  = 5.287e+00 (Chemistry minimum timestep).
     1.000e-02     5.287e+00     AriaRegion
Transient Time_Block_2: dt = 5.287
-------------------------------------------------
Transient Time_Block_2, step 169, time 2.1985e+03, time step 5.2875e+00, 61.06% complete
-------------------------------------------------
Advance AriaRegion, time 2198, time step 5.287

Memory Usage: current = 209186816 (199.5 M), high-water-mark = 217268224 (207.2 M)

Equation System AriaRegion->main:
 * Step  : Transient, Strategy: NEWTON, Time: 2.20e+03, Step: 5.29e+00
 * Matrix: Solver: "Solve_Temperature", Unknowns: 1360, Nonzeros: 11578
 * Mesh  : Processor 0 of 1: 1253 of 1253 elems, 1363 of 1363 nodes
 * Computing View Factors for enclosure space
  N O N L I N E A R               L I N E A R
```

```
---------------------   ------------------------------------
Step  Resid    Delta     Itns Status  Resid    Asm/Slv Time
---- -------- --------   ---- ------ -------- ---------------
1    6.37e-02 1.10e+01   59    ok    8.23e-07 7.0e-03/2.0e-03
2    1.10e-04 8.18e-03   58    ok    1.19e-09 7.0e-03/2.0e-03
3    3.06e-07 8.81e-05   61    ok    3.68e-12 6.0e-03/2.0e-03
Termination reason: 8.80515e-05 < nonlinear_correction_tolerance(0.001)

Field            min     @ id    max     @ id max-chg  @ id pred-err  pred-min  @ id  pred-max  @ id
------------ ---------- ---- ---------- ---- -------- ---- -------- ---------- ---- ---------- ----
* TEMPERATURE 3.8650e+02  412 8.4155e+02  512 3.93e+00 1443 3.53e-04 3.8650e+02  412 8.4155e+02  512

DT_TEMPERATURE : MIN(value,loc) = (-1.18e+00, 1443) : MAX(value,loc) = (9.68e-01, 249)
Segregated solution procedure converged after 1 iterations.
-------------- For equation system name: main --------------
Global predictor error = 3.531e-04
Time step selection: dt <= 2.326e+01 (based on Predictor-Corrector Tolerance).
Time step selection: dt <= 1.057e+01 (based on Maximum Time Step Size Ratio).
Time step selection: dt <= 6.000e+01 (based on Maximum Time Step Size).
Time step selection: dt  = 5.277e+00 (Minimum Chemistry timestep block_6).
Time step selection: dt <= 1.277e+01 (based on Stability limit for dt Ratio).
Time step selection: dt  = 1.057e+01 (Adaptive time stepping result).
Killed  1 elements this timestep, for a total of 87 dead.
        Global Variable               Value
------------------------------- ----------------
Conv_ipo                        189.224
Conv_ipo_surface_block_2_edge2_2 0
Conv_ipo_surface_block_3_edge2_2 0
killed_elements                 1
space_area                      0.155955
Tmax                            386.498 841.553
total_dead_elements             87
Region::execute() time for AriaRegion: 3.537e+00 sec.
```

27.3.1 Current Temporal Information

The first part of the above section deals with the time step and temporal information about the running model. This section can be seen below:

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
Minimum Time Step Selection:
        min           max
Time step selection: dt  = 5.287e+00 (Chemistry minimum timestep).
     1.000e-02     5.287e+00     AriaRegion
Transient Time_Block_2: dt = 5.287
------------------------------------------------
Transient Time_Block_2, step 169, time 2.1985e+03, time step 5.2875e+00, 61.06% complete
------------------------------------------------
Advance AriaRegion, time 2198, time step 5.287
```

Aria determined the current time step size based on factors from the previous time step. Here, the maximum allowed time step size is 5.287, which is based on the minimum time step allowed for the chemistry model. It can also be seen that the run is currently in `Time_Block_2` as is set up in Solution Control.

The information contained within the dotted lines allows one to gauge how far along a simulation is. In that one line, one can find the current time block, the iteration number, current simulation time, current time step, and the percent completion of the current time block.

27.3.2 Memory Reporting

One factor that determines the efficiency of a simulation is the amount of memory that is utilized. Each Sierra application will periodically output the current memory utilization - an example of this is shown below:

```
Memory Usage: current = 209186816 (199.5 M), high-water-mark = 217268224 (207.2 M)
```

One can see that at this time, 199.5 MB is currently being utilized, with a historical maximum of 207.2 MB during the course of the simulation.

27.3.3 Linear/Nonlinear Solves

The next part of the runtime information deals with the linear and nonlinear solution:

```
Equation System AriaRegion->main:
 * Step  : Transient, Strategy: NEWTON, Time: 2.20e+03, Step: 5.29e+00
 * Matrix: Solver: "Solve_Temperature", Unknowns: 1360, Nonzeros: 11578
 * Mesh  : Processor 0 of 1: 1253 of 1253 elems, 1363 of 1363 nodes
 * Computing View Factors for enclosure space
  N O N L I N E A R               L I N E A R
----------------------   ------------------------------------
Step  Resid    Delta     Itns Status  Resid    Asm/Slv Time
---- -------- --------   ---- ------ -------- ---------------
1    6.37e-02 1.10e+01   59    ok    8.23e-07 7.0e-03/2.0e-03
2    1.10e-04 8.18e-03   58    ok    1.19e-09 7.0e-03/2.0e-03
3    3.06e-07 8.81e-05   61    ok    3.68e-12 6.0e-03/2.0e-03
Termination reason: 8.80515e-05 < nonlinear_correction_tolerance(0.001)

Field            min     @ id    max     @ id max-chg  @ id pred-err  pred-min  @ id  pred-max  @ id
------------ ---------- ---- ---------- ---- -------- ---- -------- ---------- ---- ---------- ----
* TEMPERATURE 3.8650e+02  412 8.4155e+02  512 3.93e+00 1443 3.53e-04 3.8650e+02  412 8.4155e+02  512

DT_TEMPERATURE : MIN(value,loc) = (-1.18e+00, 1443) : MAX(value,loc) = (9.68e-01, 249)
Segregated solution procedure converged after 1 iterations.
```

Each bullet point here gives some information as to the solution time and strategy, as well as some information about the matrix being solved. The line `Computing View Factors ...` indicates that enclosure radiation viewfactors are being solved - more information about enclosure radiation diagnostics can be found in Section 27.3.6.

Each column of the `NONLINEAR/LINEAR` table deserves some explanation.

`Step` The nonlinear step being taken - here there are three nonlinear steps in the timestep

`Resid` Nonlinear residual at the end of each nonlinear step and before the linear steps are taken

`Delta`	Difference in the solution variable(s) compared to the last nonlinear solution
`Itns`	Number of linear iterations per nonlinear step
`Status`	Whether linear solver completed successfully (output is either `ok` or `fail`)
`Resid`	Linear residual at the end of the linear iterations
`Asm/Slv Time`	Linear system assembly/solve times

A more complete description of this table is included in the Nonlinear Solution Specifications chapter refch:nlsolv.

As an example of how to interpret the output from this table, nonlinear step 2 will be translated to text: "During nonlinear step 2, the iterative linear solver took 7.0ms to assemble and 2.0ms to solve the linear system. It completed the solve in 58 iterations and exited with a linear residual of `1.19e-09`. The nonlinear residual at the end of the nonlinear step was `1.10e-04`, which was a change of `8.18e-03` from the previous nonlinear step." This section of the log file also shows the reason why the time step exited - usually due to reaching some desired tolerance or failure to converge. Here, the solution was reached when the change in temperature from the previous nonlinear step (`Delta = 8.81e-05`) was less than the nonlinear correction tolerance of 0.001. For more information on what linear and nonlinear residuals mean, refer to the Aria Theory Manual for how the equations and linear matrix are solved.

The lines below the table also show useful output on the fields being solved - here, for temperature. Here, was can see that the minimum temperature is 386.5 at node 412, and the maximum temperature is 841.6 at node 512. Information about the predictor-corrector errors are also shown, as well as the time derivative of temperature.

27.3.4 Timestep Determination

Based on the solution at the current step, the timestep is re-calculated. This is shown in the following lines:

```
-------------- For equation system name: main --------------
Global predictor error = 3.531e-04
Time step selection: dt <= 2.326e+01 (based on Predictor-Corrector Tolerance).
Time step selection: dt <= 1.057e+01 (based on Maximum Time Step Size Ratio).
Time step selection: dt <= 6.000e+01 (based on Maximum Time Step Size).
Time step selection: dt  = 5.277e+00 (Minimum Chemistry timestep block_6).
Time step selection: dt <= 1.277e+01 (based on Stability limit for dt Ratio).
Time step selection: dt  = 1.057e+01 (Adaptive time stepping result).
```

These restrictions will change based on the method in which the solution is solved as well as the physics being solved. Here, the timestep may be limited by adaptive time stepping restrictions (predictor-corrector tolerance, maximum timestep ratio, and maximum time step size), chemistry timestep, and BDF2 stability limit. To put this block in sentence format, "The next timestep must be: less than or equal to 23.3s based on the predictor-corrector tolerance, less than or equal to 10.57s based on the maximum timestep ratio, less than or equal to 60s based on the maximum time step size, equal to 5.28s based on the ChemEQ solution, and less than or equal to 12.77s based on timestep stability." After looking at it in this manner, one can see that the Chemistry timestep of 5.28s is the limiting timestep, and this will be the timestep size which is used in the next timestep, similar to what was explained in Section 27.3.1.

27.3.5 Global Variable Output

At the end of each timestep, a summary is given of the global variables in the model. If there are global variables that the user does not wish to output the "GLOBAL VARIABLES EXCLUDED FROM LOG FILE = [list of variables]" input deck line may be used.

```
Killed  1 elements this timestep, for a total of 87 dead.
        Global Variable               Value
------------------------------ ---------------
Conv_ipo                       189.224
Conv_ipo_surface_block_2_edge2_2 0
Conv_ipo_surface_block_3_edge2_2 0
killed_elements                1
space_area                     0.155955
Tmax                           386.498 841.553
total_dead_elements            87
Region::execute() time for AriaRegion: 3.537e+00 sec.
```

This section is useful for run-time monitoring of key values, as well as how long the execution time was, in order to locate any slow-running parts of the model through time.

27.3.6 Chaparral Output - Enclosure Radiation

When enclosure radiation is defined and set up in the input file, an external library called Chaparral is utilized. Chaparral output is not automatically sent to the Sierra log file, but rather is sent to standard output (on Sandia HPC platforms, the slurm output file). Chaparral will output useful information regarding the viewfactor calculation, matrix smoothing, and the radiosity solver. Each section below will describe the output one may expect from Chaparral.

Instantiation

This output only occurs at the very start of the Chaparral output, but can help a user remember the number of processors used and the number of enclosures in the model:

```
 ********************************************************************
    C H A P A R R A L  --  Version 3.3.1 development  --  unreleased
 ********************************************************************

  Initializing for:
     number of processors  = 1
     number of enclosures  = 1
     max number of patches = 1000
```

Viewfactor Calculation

At the beginning of the viewfactor calculation, a banner will be displayed which shows the current simulation step and time, which is useful in correlating the Chaparral output to Aria output, as well as an estimate of the rowsum for each enclosure:

```
*****************************************************************
 C O M P U T E    V I E W F A C T O R S
 Step = 214   Time = 2425.53
*****************************************************************
```

The next section actually deals with the calculation of the radiation enclosure viewfactors:

```
*****************************************************************
 V I E W F A C T O R    C A L C U L A T I O N
*****************************************************************

 Calculating viewfactors for enclosure <space>
   enclosure geometry:    axisymmetric
   enclosure type:        partial (area=100), blocking
   # of rotations:        16
   # of patches:          62
   # of facets:           976
   # of nodes:            63
   spatial tolerance:     1e-06
   BSP target max depth:  10
   BSP target min length: 50
   BSP Tree num leafs:    28
   BSP Tree max depth:    5
   BSP Tree min length:   27
   BSP Tree max length:   45
   output level:          2

 Data segment memory size = 0.00Mb

 Calling VF_Hemicube()...
   resolution       = 400
   max subdivisions = 4
   min separation   = 5

 Minimum Partial Enclosure Area = 0.151296

   Minimum effective surface radius    = 0.00916667
   Minimum separation distance         = 0.00558985
   Maximum desired surface subdivision = 9, 46
   Actual maximum surface subdivision  = 4

 Data segment memory size = 0.00Mb

 Elapsed time = 3.92
```

While some of this output is merely a summary of user-specified or default tolerances, some information is useful during the simulation or for debugging. First, the user-defined name of the enclosure is shown in < > brackets (i.e., `space`). The number of facets and nodes that participate in the enclosure can also be useful in determining if the enclosure is complete in some instances. Last, if one is specifying a partial enclosure, then the `Minimum Partial Enclosure Area` is useful in correct specification of the corresponding parameter in the input file.

For simulations in which the enclosure does not change through time, the viewfactor calculation and corresponding text output will only appear once per enclosure after instantiation.

Viewfactor Smoothing

If viewfactor smoothing is specified, information about this step is also output. An example is shown below:

```
*********************************************************************
 V I E W F A C T O R    M A T R I X    S M O O T H I N G
*********************************************************************

 Smoothing of viewfactor matrix for enclosure <space>
   PCG Solver, wt = 2.0
   Max iterations = 500
   Tolerance      = 1e-07

   Enforcing reciprocity by addition...
     Nonzero lower triangular entries = 688 changed to 688
     Nonzero upper triangular entries = 687 changed to 688
     Elapsed time                     = 0.00
   Number of passes     = 1
   Number of iterations = 26
   Elapsed time         = 0.00
```

Again, the enclosure name is denoted within the < > brackets, with another summary of defined values. This section is useful for debugging problematic enclosures, since viewfactor smoothing can fail when the enclosure is ill-defined.

Viewfactor Summary

Once the viewfactors have been computed and smoothing has been applied, Chaparral will output the summary of how well the operations have done. This is shown in the `Viewfactor Matrix Summary`:

```
*********************************************************************
 V I E W F A C T O R    M A T R I X    S U M M A R Y
*********************************************************************

 Viewfactor matrix summary for enclosure <space>

 Target rowsum                 = 62
 Raw rowsum total              = 62.000000
 Raw rowsum error min          = -1.1479e-07
 Raw rowsum error max          =  1.1025e-07
 Raw rowsum error mean         =  4.9383e-09 +/- 4.9314e-08
 Smoothed rowsum total         = 62.000000
 Smoothed rowsum error min     = -7.0897e-08
 Smoothed rowsum error max     =  5.9255e-08
 Smoothed rowsum error mean    =  2.8875e-10 +/- 2.7300e-14

 Viewfactor matrix is 36.19% dense

 Total elapsed time = 3.93 (sec)
   (Initialization) = 0.01
   (Calculation)    = 3.92
   (Smoothing)      = 0.00
```

```
Data segment memory size  = 0.00Mb
```

Here one can see that for the space enclosure, the computed rowsum closely matched the target rowsum, with and without smoothing. If one is also concerned about timing, an itemized timing summary is given for the viewfactor matrix setup. This is probably the most useful section to look at if one is concerned about issues with certain enclosures, as well as judging the cost and usefulness of viewfactor smoothing.

Radiosity Solution

After the viewfactor matrix has been populated, Chaparral will use this to compute the radiosity matrix. Output for one iteration of the radiosity matrix solver is shown below:

```
*********************************************************************
 R A D I O S I T Y    S O L V E S
 Step = 214   Time = 2425.53 Time_Step = 6.18025 Iteration = 1
*********************************************************************


************************************************************************
 R A D I O S I T Y    M A T R I X    S O L V E R
************************************************************************

 Solving radiosity equations with GMRES for enclosureID <space>

 Initial residual = 1.553372e+03    tol = 2.000000e-07
    iter:    0   residual = 1.553372e+03
    iter:    1   residual = 2.657621e+04
    iter:    2   residual = 6.553812e+03
    iter:    3   residual = 1.226445e+03
    iter:    4   residual = 3.526103e+02
    iter:    5   residual = 8.437261e+01
    iter:    6   residual = 3.212877e+01
    iter:    7   residual = 8.262992e+00
    iter:    8   residual = 2.397060e+00
    iter:    9   residual = 3.692025e-01
    iter:   10   residual = 1.010501e-01
    iter:   11   residual = 3.528043e-02
    iter:   12   residual = 6.867475e-03
    iter:   13   residual = 1.206750e-03
    iter:   14   residual = 9.768426e-05
    iter:   15   residual = 1.063625e-05
    iter:   16   residual = 1.902986e-06
    iter:   17   residual = 3.433117e-07
    iter:   18   residual = 9.218341e-08

    Real residual is  1.48685e-09
 Computing final fluxes

 Total iterations:    18
 Total elapsed time:  0.00 (sec.)
   GMRES setup time:  0.00 (sec.)
```

```
  GMRES solve time:  0.00 (sec.)
Residual L2 norm:    1.48685e-09
Flux Integration:    -0.219045
Minimum Emissivity:  0.3
Maximum Emissivity:  1
Minimum Temperature: 443.752, Element: 778
Maximum Temperature: 1200, Element: 415
```

This output is somewhat similar to the Aria run-time output, in that the linear iteration count is given along with the linear residual, though the Chaparral output is more detailed, as it gives the linear residual at each linear step instead of a summary. Minimum and maximum values of emissivity and temperature in the enclosure can also help in understanding the behavior of the radiosity solve through time.

27.4 Run Summary

At the end of a completed run, a summary is printed to the log file, which starts with something along the lines of the following:

```
Procedure AriaProcedure complete                              May 14 2014 11:19:44
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
Execution complete
```

The different sections of the run summary will be described below.

27.4.1 Timing

The Sierra code as well as Aria contain many timers which encapsulate certain operations of each simulation. The timing summary shows the results of these timers and are organized so that aspects of the model that take more time can be found. A sample timing summary can be seen below and will be further explained:

```
Timing summary of 1 processor
                 Timer                    Count       CPU Time             Wall Time
---------------------------------------- ----- -------------------- --------------------
Sierra                                       1    15:14.415 (100.0%)    15:19.072 (100.0%)
  Procedure AriaProcedure
    Initialize                               1        0.017 (<0.01%)        0.017 (<0.01%)
    Execute                                  1    15:13.010 (99.85%)    15:17.659 (99.85%)
    Mesh input                               1        0.168 ( 0.02%)        0.170 ( 0.02%)
    Mesh output                            357        0.905 ( 0.10%)        1.196 ( 0.13%)
    Region AriaRegion
      Initialize                             1        0.015 (<0.01%)        0.017 (<0.01%)
      Execute                              357    15:11.364 (99.67%)    15:15.619 (99.62%)
        Nonlinear Iteration                997       16.482 ( 1.80%)       16.568 ( 1.80%)
          Preprocessing                    997        6.811 ( 0.74%)        6.865 ( 0.75%)
            Chemistry Kinetics             997        6.768 ( 0.74%)        6.820 ( 0.74%)
          LinSys Assembly                  997        6.091 ( 0.67%)        6.119 ( 0.67%)
          LinSys Solve                     997        2.458 ( 0.27%)        2.472 ( 0.27%)
          LinSys Scatter                   997        0.276 ( 0.03%)        0.273 ( 0.03%)
          Nonlinear Utilities             3988        0.668 ( 0.07%)        0.669 ( 0.07%)
```

```
      Postprocessing                    358        0.034 (<0.01%)        0.046 (<0.01%)
      Utilities                        2144    14:51.790 (97.53%)    14:54.651 (97.34%)
        Parallel Sync                  5776        0.012 (<0.01%)        0.010 (<0.01%)
        Chaparral Viewfactor            357    14:51.546 (97.50%)    14:54.417 (97.32%)
        Chaparral Radiosity             997        0.629 ( 0.07%)        0.623 ( 0.07%)
      Chemistry Utilities              3062        0.085 (<0.01%)        0.079 (<0.01%)
      Contact Search                    357        0.003 (<0.01%)        0.000 (<0.01%)
                                       1074        0.259 ( 0.03%)        0.372 ( 0.04%)
      Tmax                              358        0.275 ( 0.03%)        0.277 ( 0.03%)
    Mesh input                            1        0.168 ( 0.02%)        0.170 ( 0.02%)
    Mesh output                         357        0.904 ( 0.10%)        1.196 ( 0.13%)
      Results output                    358        0.326 ( 0.04%)        0.420 ( 0.05%)
      Restart output                    358        0.549 ( 0.06%)        0.729 ( 0.08%)
      History output                    358        0.015 (<0.01%)        0.027 (<0.01%)
      Heartbeat output                  358        0.020 (<0.01%)        0.036 (<0.01%)
      Control                           109        0.001 (<0.01%)        0.005 (<0.01%)
      Vis output                        358        0.001 (<0.01%)        0.000 (<0.01%)
Perf: RunSierra                           1        1.218 ( 0.13%)        1.225 ( 0.13%)
  Perf: RunSierra::parse                  1        1.218 ( 0.13%)        1.225 ( 0.13%)
  Perf: RunSierra::Domain::execute        1    15:13.197 (99.87%)    15:17.846 (99.87%)
```

Each entry shows the CPU and Wall Times associated with it, along with a percentage, which is the percentage of the total run time. One can also see where a timer resides with respect to the Procedure, Region, etc. by the level of indention in the timing summary. For example, one can see by the indention that `Nonlinear Iteration` is a part of `Execute`, which resides in the `Region ariaRegion`. The sum of timers of all entries on a certain indention level should be less than or equal to the timer of the parent entry, since the parent timer is a wrapper around the children timers and the corresponding code execution. For example, `Preprocessing`, `LinSys Assembly`, `LinSys Solve`, `LinSys Scatter`, and `Nonlinear Utilities` timers are equal to 16.304s of CPU time, which is less than the 16.482s of `Nonlinear Iteration`.

Here is a general description of what each timer means:

`Sierra`	Complete encapsulation of the simulation
`Procedure`	Procedure-level of the simulation, including Solution Control and Regions
`Initialize`	Initialization of the Procedure
`Execute`	Execution time for the procedure; in a simple one-region simulation, this will be very similar to Region `Execute`
`Mesh Input`	Time to read in the input Genesis mesh
`Mesh Output`	Time required to output any results, e.g., Results, Restart, History, Heartbeat Output
`Region`	Region scope of the simulation, including physics setup
`Initialize`	Initialization of the Region
`Execute`	Execution time for the Region
`Nonlinear Iteration`	Performance of the nonlinear solves
`Preprocessing`	Setup for the nonlinear iterations
`Chemistry Kinetics`	Setup of chemistry volume source and ODE solver

`LinSys Assembly`	Assembly of the Linear system of matrices
`LinSys Solve`	Solution of the linear system of matrices
`LinSys Scatter`	Division of the linear system of matrices in parallel
`Nonlinear Utilities`	A combination of several utilities (e.g., chemistry, radiation) that occur during the nonlinear step
`Postprocessing`	Execution of user-requested post-processing fields
`Utilities`	Wrapper around utilities required for system of equations
`Parallel Sync`	Communication required between processors for simulations run in parallel
`Chaparral Viewfactor`	Viewfactor calculation for enclosure radiation
`Chaparral Radiosity`	Radiosity calculation for enclosure radiation
`Chemistry Utilities`	Chemistry solve which occurs outside of the nonlinear loop
`Contact Search`	Search required to find contact interactions
`Tmax`	Determination via Encore of Tmax postprocessor
`Mesh input`	Time to read in the Genesis mesh
`Mesh output`	Time to output results
`Results output`	Time to output Results Output block(s)
`Restart output`	Time to output Restart Output block(s)
`History output`	Time to output History Output block(s)
`Heartbeat output`	Time to output Heartbeat Output block(s)
`Control`	Determination of when to output
`Vis output`	Time to output in-situ visualization output
`Perf:`	Sierra performance-related timers

It can be seen that in this simulation, the bulk of the time is spent in `Chaparral Viewfactor`. This is due to the fact that this simulation involves element death, and the radiation enclosure is updated each time that elements are killed.

27.4.2 Memory

In addition to the run-time memory statistics, there is a memory summary printed at the end of the log file. An example of this is shown below:

```
Memory summary of 1 processor
           Metric             Largest      Processor     Smallest      Processor
---------------------------- --------- ----------------- --------- -----------------
Dynamically Allocated (Heap)   73.8 MB on 0 at 13:33.348   73.8 MB on 0 at 13:33.348
Largest Free Fragment (Heap)   43.0 MB on 0 at 13:33.348   43.0 MB on 0 at 13:33.348
         Total Memory In Use  398.1 MB on 0               398.1 MB on 0
           Major Page Faults    0 flts on 0                 0 flts on 0
```

The output of this memory summary becomes more varied for a parallel simulation, where the model may not be evenly distributed amongst processors.

Another useful piece of information related to memory reporting is contained in the `Performance metric summary`:

```
Avg Available memory per processor 4036.1 MB
```

Here one can see the simulation memory high-water mark, as well as the available memory per processor on the system on which the simulation was run. This information becomes important when one would like to maximize the amount of memory is used, which can reduce the number of processors required. For example, in a large series of uncertainty quantification simulations, reduction of processor count can increase throughput on HPC platforms with queuing systems.

27.5 Summary

Though it can be difficult to interpret the sometimes large amounts of data that are output via log files, this information can be useful in ascertaining a simulation's performance and can also be used to improve the manner in which it is run. Though some of the information presented here is Aria-specific, much of it is also applicable when using other Sierra applications and can aid in use of those codes as well.

Chapter 28

Diagnostic Output

28.1 About Diagnostic Streams

Aria is instrumented with a diagnostic output capability that can be very useful when you're debugging problems. You can tell Aria to write additional information to the log file by enabling different *print masks*. The list of available print masks is summarized in table 28.1; this list is also supplied in the Aria runtime help which is printed if you add `-h` to your `sierra` command line.

There are two ways to activate a print mask for diagnostic printing. The first is to use the `-arialog` command line option. When you're using the `sierra` script this is a little odd because you have to pass that option as an option to the script. By example, if you want to enable the sensitivity checker you can do that like this

```
sierra aria ...-arialog sens_check
```

where `...` all of the other command line options you typically use.

The second way to enable a print mask is more versatile and is done through your aria input file. At the highest level of the input file – somewhere in the `BEGIN SIERRA` block – you can add a `DIAGNOSTIC CONTROL` block. This enables you to specify intervals in time or time step number over which print masks should be enabled. As a simple example, let's say you want to enable the sensitivity checker (see `sens_check` in 28.1) around time step number 47. Here's an example input file snippet that allows you to do that:

```
Begin Diagnostic Control arialog
   From Step 46 to 48 enable sens_check
End
```

With this, the sensitivity checker will only be active for those three time steps (it's also valid to say `From Step 47 to 47 ...`). The end of this chapter includes a detailed command reference for that input.

Lastly, I'll just mention here for now that there are additional capabilities for logging, such as per-processor log files (be default only processor zero logs output). If you need more advanced diagnostics contact the Aria developers for help.

28.2 Diagnostic Output Command Reference

28.3 Diagnostic Control

Scope: Sierra

Print Mask	Description
bc	Display boundary condition information
debug	Display debug diagnostic information
eq	Display equation information
expression	Display expression information
hadapt	Display h-adapt diagnostic information
finite_check	Display warning messages about any non-finite expression values or sensitivities
nonlinear	Display nonlinear solver information
pp	Display postprocessor diagnostic information
sens_check	Display messages generated by expression sensitivity checker
fast_sens_check	Display messages generated by sensitivity checker (skipping FAD, element, and kernel expressions)
species	Display species information
transfer	Display transfer information
dualsolve	Display dual solve information
plugin	Display plugin information
chaparral	Display chaparral information
utility	Display equation system utility information

Table 28.1. Aria diagnostic output print masks.

```
Begin Diagnostic Control Name

    Enable Printmask

    From Step StartStep To EndStep Enable Printmask

    Set Information Stream Path File_path
End
```

Summary

Specifies the diagnostic writer to set output control features.

Description

The diagnostic and informational output can be selectively enabled based on time, step or an application specified condition. During the application's procedure execution loop, the diagnostic controller evaluates the enclosed line commands in the order specified in the input deck. The diagnostic options specified in the first line command that meets its criteria are applied.

Since control parameters are only applied when the criteria is met, it is important to include an ENABLE line command with the base settings to be applied as a baseline. Refer the the '-h' output for a complete list of diagnostic writers and available values.

fmwkout Specify a comma separated list of: contact Display contact diagnostic information coverage Collect and display traceable function usage coverage dump-load Dump domain atfer mesh load dump-setup Dump domain after setup input-check Check input deck and mesh load, does not execute members Display data structure members messages parameters Display parameter diagnostic information scontrol Display solver control diagnostic information search Display search diagnostic information syntax-check Check syntax of input deck, does not load mesh or execute trace Display execution trace trace-stats Display execution time and memory usage during trace transfer Display transfer diagnostic information warning Display warning messages

iossdebug Specify a comma separated list of: coverage Collect and display traceable function usage coverage error Display error messages members Display data structure members

messages mesh Display mesh I/O diagnostic information restart Display restart I/O diagnostic information results Display results I/O diagnostic information serialized Display serialized I/O information trace Display execution trace trace-stats Display execution time and memory usage during trace warning Display warning messages

prsrdebug Specify a comma separated list of: coverage Collect and display traceable function usage coverage error Display error messages members Display data structure members messages parser Display parser diagnostic information trace Display execution trace trace-stats Display execution time and memory usage during trace warning Display warning messages

The options which may be enabled varies for each applicaition, diagnostic and information writer. To obtain a list of available options, use the sierra app -i -O -h command.

See Diagnostic Stream for specifying the output destination.

28.3.1 Enable

Scope: Diagnostic Control

`Enable` *`Printmask`*

Parameter	Value	Default
Printmask	`"string"`	`undefined`

Summary Specifies the options to enable when no other control option criteria are satisfied.

28.3.2 From Step

Scope: Diagnostic Control

`From Step` *`StartStep`* `To` *`EndStep`* `Enable` *`Printmask`*

Parameter	Value	Default
StartStep	`integer`	`undefined`
EndStep	`integer`	`undefined`
Printmask	`"string"`	`undefined`

Summary Specifies the options to enable when the step is within the specified range.

28.3.3 Set Information Stream Path

Scope: Diagnostic Control

`Set Information Stream Path` *`File_path`*

Parameter	Value	Default
File_path	`string`	`undefined`

Summary File path to information stream information to.

28.3.4 Diagnostic Stream

Scope:

Diagnostic Stream *File_name* [*Indent-streambuf-flags*$_1$[*Indent-streambuf-flags*$_2$]]

Parameter	Value	Default
File_name	string	undefined

Summary File path to write diagnostic messages to.

28.3.5 Enable

Scope:

Enable *Printmask*

Parameter	Value	Default
Printmask	"string"	undefined

Summary Specifies the options to enable when no other control option criteria are satisfied.

28.3.6 From Step

Scope:

From Step *StartStep* To *EndStep* Enable *Printmask*

Parameter	Value	Default
StartStep	integer	undefined
EndStep	integer	undefined
Printmask	"string"	undefined

Summary Specifies the options to enable when the step is within the specified range.

28.3.7 Set Information Stream Path

Scope:

Set Information Stream Path *File_path*

Parameter	Value	Default
File_path	string	undefined

Summary File path to information stream information to.

Chapter 29

Level Set Reference

29.1 Level Set Overview

Aria employs a level set capability supported by the Krino library. Here the basic Krino capability is used to define the level set interface and the problem physics is handled with equations native to Aria.

To use this capability one would define Level_Set for the DOF on a particular volume mesh entity (block) as described in 6.5. Then the level set interface would be defined using the LEVEL SET INTERFACE commands. In tracking the level set interface the Narrow Band Width is the distance over which the level set distance is calculated. Outside this narrow band, the distance is clipped when the redistancing calculation is performed. Advection algorithms in Aria apply over the entire domain, but the redistancing algorithms in Krino support this clipping. Tailoring the Narrow Band Width to one's problem can help reduce the cost of redistancing and reduce solution artifacts away from the interface.

29.2 Level Set Interface

Scope: Aria Region

```
Begin Level Set Interface InterfaceName

    Composite Name {=|are|is} Variable
    Distance Variable {=|are|is} Variable
    Extension Velocity {=|are|is} Value...
    Initial Offset Distance {=|are|is} Offset
    Initial Scale Factor {=|are|is} Scale
    Max Feature Size On Surfaces {=|are|is} Value
    Narrow Band Element Size Multiplier {=|are|is} Value
    Narrow Band Width {=|are|is} Value
    Perform Initial Redistance {=|are|is} performInitialRedistance
    Redistance Method {=|are|is} RedistanceMethodType
    Reinitialize Every Step
    Simple Max Feature Size On Surfaces {=|are|is} Value
    Begin Analytic Initial Condition BlockName
    End

    Begin Compute_Surface_Distance SurfaceName
    End
```

```
    Begin Motion Specification Motion
    End

End
```

Summary Main block for krino level set interface control mechanics.

Description This is the main block for the level set interface control mechanics. This block specifies a level set interface that is integrated in time using an extension velocity. The plan is that the physics application code will see only this mechanics, although there may be several other supporting mechancis that are plugged in.

29.2.1 Composite Name

Scope: Level Set Interface

`Composite Name {= | are | is}` *`Variable`*

Parameter	Value	Default
Variable	`string`	`undefined`

Summary Optionally specify a composite level set name which may be used across multiple level sets to define composite phases.

29.2.2 Distance Variable

Scope: Level Set Interface

`Distance Variable {= | are | is}` *`Variable`*

Parameter	Value	Default
Variable	`string`	`undefined`

Summary Specify the distance variable name.

29.2.3 Extension Velocity

Scope: Level Set Interface

`Extension Velocity {= | are | is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary Specify the extension velocity used to evolve the level set field. Default is 0 0 0.

29.2.4 Initial Offset Distance

Scope: Level Set Interface

`Initial Offset Distance {=|are|is}` *Offset*

Parameter	Value	Default
Offset	`real`	`undefined`

Summary Specify an offset distance to be applied to the initial level set. Allows the user to grow or shrink the initialized level set.

29.2.5 Initial Scale Factor

Scope: Level Set Interface

`Initial Scale Factor {=|are|is}` *Scale*

Parameter	Value	Default
Scale	`real`	`undefined`

Summary Specify an scale factor to be applied to the initial lsvel set. Allows the user to multiple the entire level set by a constant, primarily useful for nested (concentric) level sets. This modifier gets applied after the INITIAL OFFSET DISTANCE, if there is one.

29.2.6 Max Feature Size On Surfaces

Scope: Level Set Interface

`Max Feature Size On Surfaces {=|are|is}` *Value*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify the maximum feature size on surfaces, features smaller than this will be destroyed near all surfaces. IN BETA TESTING!

29.2.7 Narrow Band Element Size Multiplier

Scope: Level Set Interface

`Narrow Band Element Size Multiplier {=|are|is}` *Value*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify the width of the narrow band width used for the level set field as the given multiplier times the maximum element length scale. If this command is omitted, the distance will be computed throughout the entire domain, but this is not generally scalable in terms of memory or cpu time.

29.2.8 Narrow Band Width

Scope: Level Set Interface

`Narrow Band Width {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify the width of the narrow band. This form is deprecated. Use NARROW BAND ELEMENT SIZE MULTIPLIER instead.

29.2.9 Perform Initial Redistance

Scope: Level Set Interface

Summary Perform redistancing operation just after intial conditions are applied.

29.2.10 Redistance Method

Scope: Level Set Interface

`Redistance Method {=|are|is}` *`RedistanceMethodType`*

Parameter	Value	Default
RedistanceMethodType	`{closest_point|fast_marching}`	`CLOSEST_POINT`

Summary Specify the algorithm to be used for redistancing.

29.2.11 Reinitialize Every Step

Scope: Level Set Interface

Summary Specify that this level set interface should be re-initialized every time step. For example this can be combined with an analytic mesh surface initial condition that is moving due to applied displacements to move the CDFEM interface location with the mesh surface.

29.2.12 Simple Max Feature Size On Surfaces

Scope: Level Set Interface

`Simple Max Feature Size On Surfaces {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify the maximum feature size on surfaces, features smaller than this will be destroyed near all surfaces (simple version) IN BETA TESTING!

29.2.13 Narrow Band Width

Scope:

`Narrow Band Width {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify the width of the narrow band. This form is deprecated. Use NARROW BAND ELEMENT SIZE MULTIPLIER instead.

29.3 Level Set Initial Conditions

As previously stated one can use the Aria initial condition definitions described in 8 by using Level_Set as the DOF.

Alternatively, one can also use the the level set initial condition support provided by the Krino library. In a multi-region analyses where one has a level set region whose level result is being transferred to another region which controls output of the results, synchronization of the initial state may become an issue. In this event the initial conditions being transferred from the Krino region can be overwritten by the region controlling output. This problem can be dealt with by orchestration of initialization by Solution Control using the command structure included below wherein LS_Region is the level set region.

If one chooses to use the initial conditions from within Aria instead, the command line for the Initialize command block is not used and the "Use Initialize" command line would not appear in the Solution Control "System" command block.

```
Begin Solution Control Description
  Use System Main
    .
    .
  Begin Initialize The_Init_Block
    Advance LS_Region
    Event LS_Region_REINITIALIZE_LEVELSETS
  End
  Begin System Main
    Use Initialize The_Init_Block
    .
    .
  End System Main
    .
    .
End Solution Control Description
```

29.4 Analytic Initial Condition

Scope: Level Set Interface

`Begin Analytic Initial Condition` *`BlockName`*

```
Composition Method {=|are|is} CompositionMethod
Cylinder P1 {=|are|is} Param2_1 Param2_2 Param2_3 P2 {=|are|is} Param5_1 Param5_2 Param5_3
Radius {=|are|is} Param8 [ Sign = Sign |Motion = Motion ]
Ellipsoid Center {=|are|is} Param1_1 Param1_2 Param1_3 Semiaxes {=|are|is} Param2_1 Param2_2
Param2_3
Facets Format {=|are|is} Facet_format Filename {=|are|is} Facet_filename [ Scale
= ScaleFactor |Sign = Sign |Motion = Motion ]
Mesh Surfacename [ Sign {=|are|is} Sign ]
Plane Normal {=|are|is} Param2_1 Param2_2 Param2_3 Offset {=|are|is} Param5 [ Sign {=
|are|is} Sign ]
Random [ Seed {=|are|is} Seed ]
Sphere Center {=|are|is} Param2_1 Param2_2 Param2_3 Radius {=|are|is} Param5 [ Sign
= Sign |Motion = Motion ]
End
```

Summary Allows the specification of analytic initial condition blocks. Must specify at least one analytic shape. Can have one to many analytic shapes in any combination on a single level set interface.

29.4.1 Composition Method

Scope: Analytic Initial Condition

`Composition Method {=|are|is}` *CompositionMethod*

Parameter	Value	Default
CompositionMethod	{maximum_signed_distance| minimum_signed_distance}	undefined

Summary Specify the method used to combine the results of the individual surface distance calculations. Using MINIMUM_SIGNED_DISTANCE, the resulting negative region will be the union of the negative regions of the individual surfaces. The positive region will be the intersection of the positive regions of the individual surfaces. Using MAXIMUM_SIGNED_DISTANCE, the resulting negative region will be the intersection of the negative regions of the individual surfaces. The positive region will be the union of the positive regions of the individual surfaces. The default is MINIMUM_SIGNED_DISTANCE.

29.4.2 Cylinder

Scope: Analytic Initial Condition

`Cylinder P1 {=|are|is}` *Param2*$_1$ *Param2*$_2$ *Param2*$_3$ `P2 {=|are|is}` *Param5*$_1$ *Param5*$_2$ *Param5*$_3$ `Radius {=|are|is}` *Param8* `[ Sign =` *Sign* `|Motion =` *Motion* `]`

Parameter	Value	Default
Param2	real_1 real_2 real_3	undefined
Param5	real_1 real_2 real_3	undefined
Param8	real	undefined
Sign	integer	undefined
Motion	string	undefined

Summary Specify cylinder as interface surface, with given radius and two endpoints of axis p1 and p2. The length of the cylinder is norm(p2 - p1).

29.4.3 Ellipsoid

Scope: Analytic Initial Condition

`Ellipsoid Center {=|are|is}` *Param1*$_1$ *Param1*$_2$ *Param1*$_3$ `Semiaxes {=|are|is}` *Param2*$_1$ *Param2*$_2$ *Param2*$_3$

Parameter	Value	Default
Param1	`real_1 real_2 real_3`	`undefined`
Param2	`real_1 real_2 real_3`	`undefined`

Summary Specify Ellisoid as interface surface with given center and semiaxes. Note the result is not an exact distance function except in the case of a sphere (equal semiaxes).

29.4.4 Facets

Scope: Analytic Initial Condition

`Facets Format {=|are|is}` *Facet_format* `Filename {=|are|is}` *Facet_filename* `[ Scale =` *ScaleFactor* `|Sign =` *Sign* `|Motion =` *Motion* `]`

Parameter	Value	Default
Facet_format	`{exo|fac|ply|stl}`	`undefined`
Facet_filename	`string`	`undefined`
ScaleFactor	`real`	`undefined`
Sign	`integer`	`undefined`
Motion	`string`	`undefined`

Summary Specify a file containing a surface from which the level set should be initialized.

29.4.5 Mesh

Scope: Analytic Initial Condition

`Mesh` *Surfacename* `[ Sign {=|are|is}` *Sign* `]`

Parameter	Value	Default
Surfacename	`string`	`undefined`

Summary Specify a mesh surface as interface surface.

29.4.6 Plane

Scope: Analytic Initial Condition

`Plane Normal {=|are|is}` *Param2*$_1$ *Param2*$_2$ *Param2*$_3$ `Offset {=|are|is}` *Param5* `[ Sign {=|are|is}` *Sign* `]`

Parameter	Value	Default
Param2	`real_1 real_2 real_3`	`undefined`
Param5	`real`	`undefined`

Summary Specify plane as interface surface with given normal and offset. Equation of plane is 0 = normal/mag(normal) dot x + offset. Given the vector r to a point on the plane, the offset is -normal/mag(normal) dot r.

29.4.7 Random

Scope: Analytic Initial Condition

Summary Specify initial condition to pseudorandom field between -1 and 1.

29.4.8 Sphere

Scope: Analytic Initial Condition

`Sphere Center {=|are|is}` *Param2*$_1$ *Param2*$_2$ *Param2*$_3$ `Radius {=|are|is}` *Param5* `[ Sign =` *Sign* `|Motion =` *Motion* `]`

Parameter	Value	Default
Param2	`real_1 real_2 real_3`	`undefined`
Param5	`real`	`undefined`
Sign	`integer`	`undefined`
Motion	`string`	`undefined`

Summary Specify sphere as interface surface with given center and radius.

Chapter 30

Local Coordinate System Reference

30.1 Local Coordinate Systems

Aria provides utilities for convenient transformation of model specific information to the computational coordinate frame. Several common local coordinate systems - rectangular, cylindrical, spherical and conical - are pre-defined in Aria.

Known Issue: It has been determined that the conical local coordinate system has not been correctly implemented. Users should avoid use of this coordinate system until this has been corrected.

Figure 30.1 shows how each of the afore-mentioned local coordinate systems are constructed using the information supplied in the LOCAL COORDINATE SYSTEM command block. As a general rule, ORIGIN defines the origin of the local coordinate system, VECTOR defines the z-axis of the local coordinate system, and POINT is used to create the x-axis of the local coordinate system. The y-axis is then constructed via a cross-product of the x- and z- axes.

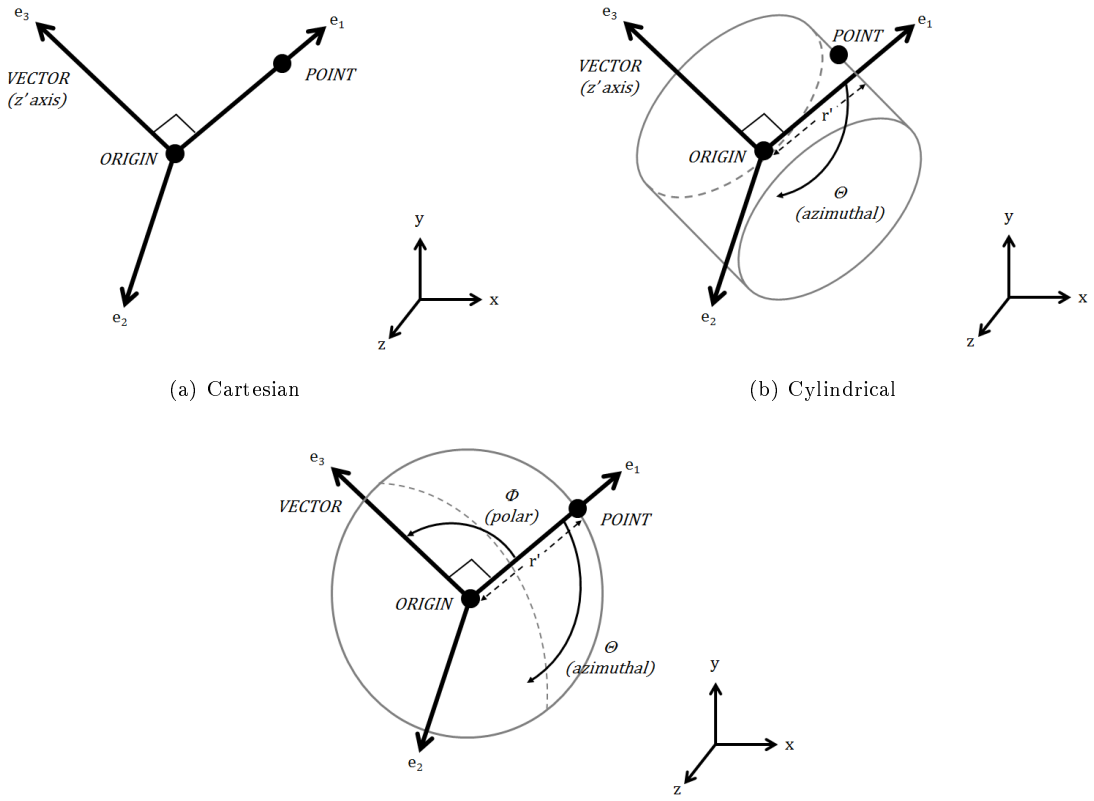
Given ORIGIN **O**, VECTOR **V**, and POINT **P** for any local coordinate system, internal initialization of geometric data is achieved via the following pseudo-code in 3D:

```
CoordinateSystem::setLocalCoordinateAxes(Origin O, Vector V, Point P)
{
     e3 = unit_vector(V)             # z axis
     D1 = P - O                      # vector from origin to to point
     D2 = projected_vector(D1, e3)   # project D1 in direction of e3
     D3 = D1 - D2                    # orthogonal component of D1 to e3
     e1 = unit_vector(D3)            # x axis
     e2 = cross_product(e3, e1)      # y axis

     # Saved variables
     origin = O                      # origin
     x_axis = e1
     y_axis = e2
     z_axis = e3
}
```

At point **P**, a local z-axis $\mathbf{e}_3$ is defined as the unit vector in the direction of vector **V**. The local x-axis $\mathbf{e}_1$ is then defined as an orthogonal projection from point **P** to $\mathbf{e}_3$ and finally, the local y-axis $\mathbf{e}_2$ is defined based on the cross product of $\mathbf{e}_3$ and $\mathbf{e}_1$.

Classical transformations consider the evaluation of vector or tensor components from one basis, $\mathbf{e}_i$, in an



(a) Cartesian

(b) Cylindrical

(c) Spherical

Figure 30.1. A depiction of the available local coordinate system options and how they are constructed given ORIGIN, VECTOR, and POINT.

alternative basis $\mathbf{e}'_i$ By supplying parameters for the pre-defined coordinate system in the LOCAL COORDINATE SYSTEM command block one in effect defines the orientation of the local coordinate system block relative to the computational coordinate directions. Using this information any appropriate transformations can be computed internal to the code. Finally, each local coordinate system can be associated with multiple element blocks using the LOCAL COORDINATE SYSTEM line command.

Local coordinate systems are particularly useful for tensor material properties defined using their local principal values. In this case one must select a "generalized" material model in order to fully utilize the principal component values. Current support of this feature is currently limited to material diffusive coefficients - however, further support is planned.

Given any point $\mathbf{P}$ in space at which the global value of a given tensor based material property $[\mathbf{K}]$ is to be evaluated, the local principal axes based on the initialization described previously must be computed. This can be represented for the various local coordinate systems by the following pseudo-code:

```
CartesianCoordinateSystem::getPrincipalAxes(Point P, Vector e1, Vector e2, Vector e3)
{
     e1 = x_axis                    # principal x axis
     e2 = y_axis                    # principal y axis
     e3 = z_axis                    # principal z axis
```

```
}

CylindricalCoordinateSystem::getPrincipalAxes(Point P, Vector e1, Vector e2, Vector e3)
{
     e3 = z_axis                      # principal z axis = same as LCS z axis
     D1 = P - origin                  # vector from origin to to point
     D2 = projected_vector(D1, e3)    # project D1 in direction of e3
     D3 = D1 - D2                     # orthogonal component of D1 to e3
     e1 = unit_vector(D3)             # principal x axis in r
     e2 = cross_product(e3, e1)       # principal y axis in theta
}

SphericalCoordinateSystem::getPrincipalAxes(Point P, Vector e1, Vector e2, Vector e3)
{
     D1 = P - origin                  # vector from origin to to point
     e1 = unit_vector(D1)             # principal x axis in r
     e2 = cross_product(z_axis, e1)   # principal y axis in theta
     e3 = cross_product(e1, e2)       # principal z axis in phi
}
```

If the principal values K_{11}, K_{22} and K_{33} are provided, these correspond to principal directions $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ respectively. Given $[\mathbf{K}]$ defined locally in the local coordinate system, the transformation matrix $[\mathbf{M}]$ that rotates the orthogonal triplet $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ into the global coordinate space $\{x, y, z\}$ is computed and the material property in global coordinates is defined as

$$[\mathbf{K}_g] = [\mathbf{M}]^T [\mathbf{K}] [\mathbf{M}] \tag{30.1}$$

30.2 Local Coordinate System

Scope: Sierra

```
Begin Local Coordinate System Coordinate_System_Name
    Origin {=|are|is} Value...
    Point {=|are|is} Value...
    Type {=|are|is} CoordType
    Vector {=|are|is} Value...
End
```

Summary — Defines the local coordinate system parameters.

Description — This Command Block allows the definition of a coordinate system that can be used to transform values for output or field calculations. For any specific local coordinate system three pieces of information are required for completely defining a coordinate system are the origin, a coordinate axis vector and a point lying on one of the other axes. Using this information a transformation from the local coordinate system to the computational coordinate system can be determined.

30.2.1 Origin

Scope: Local Coordinate System

`Origin {= | are | is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary — Local coordinate system origin.

Description — Specifies the point as the origin of a local coordinate system. The number of coordinates may be either 2 or 3 depending on the spatial dimension. By default, a value of z=0.0 is used for 2D.

30.2.2 Point

Scope: Local Coordinate System

`Point {= | are | is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary — Local coordinate alternative point.

Description — Declare the point that lies on an alternate axis and is in the plane subtended by the specified VECTOR and the vector defined by the point and the origin.

30.2.3 Type

Scope: Local Coordinate System

`Type {= | are | is}` *`CoordType`*

Parameter	Value	Default
CoordType	`{cartesian | conical | cylindrical | spherical}`	`undefined`

Summary — Local coordinate system type.

Description — This line command selects the coordinate system type.

30.2.4 Vector

Scope: Local Coordinate System

`Vector {= | are | is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary Local coordinate axis vector.

Description Declare the vector that defines one of the coordinate axes. The specific usage depends on the selected coordinate system type.

30.2.5 Local Coordinate System

Scope:

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

Parameter	Value	Default
Mesh Entities	`string`	`undefined`

Summary Associate coordinate system with mesh entity.

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

Chapter 31

Bulk Volume Reference

31.1 Bulk Volume Element

Beta Capability: The bulk volume capability is under active development hence the capability should be used with some caution.

For some engineering problems the variation of a field variable over the problem domain can be roughly represented by some average behavior. In these cases it may be more appropriate to understand the averaged behavior rather than to resolve the behavior in fine detail. When numerically modeling these types of problems, it is convenient to devise lumped parameter models that simulate this averaged behavior. One such modeling strategy, a bulk volume model, is employed to capture bulk fluid behavior without regard to spatial resolution. Herein the bulk volume model is described within the context of a Galerkin finite element model where the terms bulk volume element and bulk node are often used synonymously.

Following standard finite element method procedures, balance equations are written for a physical region surrounding the bulk volume where the equations possibly include contributions from unknowns associated with the bulk volume. Similarly a general conservation equation is written for a bulk volume system with flux contributions provided from bounding surfaces of the finite element method discretization 5. Under the assumption that the volume is known in a quasi-static sense, the spatial dependence is eliminated from equation terms containing temporal derivatives. The conservation equation for the bulk volume system is defined in the same manner as for other element blocks 6 except that it may be specified from a command block that specifically describes the bulk volume system. This balance relation provides closure for the discretized system of equations.

It is important to note that the bulk volume element interacts with the meshed model through boundary conditions and thus serves as an intervening media whose characterization has yet to be determined. Hence the methodology does not apply to problems in which the nature of interaction is known a priori. One example for which the interaction is known a priori would be convective heat transfer between two parallel surfaces of different fixed temperatures. Here heat transfer occurs directly between the two surfaces and need not be partitioned. In this case use of the bulk volume element would wrongfully introduce an additional resistance to flow of heat between the two surfaces.

Since the bulk volume degree-of-freedom is defined in terms of a time derivative, its transient evolution follows naturally. For steady problems not requiring solution converged parameter updates it is possible to solve for the bulk volume degree-of-freedom directly. Otherwise the steady-state solution for bulk volume degree-of-freedom must be obtained using a false transient simulation.

Usage of the bulk volume capability is outlined below.

```
Begin Sierra myJob
```

```
  .
  Begin Aria Material a_bulk_volume
   .
   properties for bulk volume

  End
  .
  Begin Procedure My_Aria_Procedure
     .
    Begin Solution Control Description
     .
     transient solution control commands
     .
    End
    .
    Begin Aria Region My_Region
     .
     Begin bulk volume bulk_volume_name
       material = a_bulk_volume
       EQ bulk_volume_name for bulk_volume_dof with P0 using mass
       .
       bulk volume specific definitions
       .
     End
     .
     other Region level commands
     .
    End
    .
  End
  .
End Sierra myJob
```

31.1.1 Boundary Conditions

At present there are two supported methods for applying boundary conditions that couple equations on a bulk volume element to volumetric equations. For heat transfer problems where the energy equation is being solved for temperature on both the volume and the bulk volume element then either a convective or a radiative boundary condition block can be coupled to the bulk volume element. For either type of boundary condition the "USE BULK ELEMENT" line command should be added to the input block, further details can be found in Chapter 33.

For other equations boundary conditions may be applied by using a line command of the form:

```
BC BULKNODE_FLUX FOR volume_equation ON surface_name = model_name BULK_NODE=bulk_node_name
```

In some cases it is desirable to couple equations being solved for different degrees of freedom on the volume and the bulk volume element. For example coupling a porous media equation for the gas phase temperature on a volume block to an overall temperature on the bulk volume element, or an equation solved for enthalpy to an equation solved for temperature. If the same degree of freedom is not present on the bulk node and the volume then the following mappings are checked for:

- Same degree of freedom (e.g. temperature, pressure, mass fraction) but without a material phase

specification on the bulk node when one is present on the volume block.

- Temperature degrees of freedom on the volume will check for enthalpy in either the same material phase or no material phase on the bulk node (and vice versa for an enthalpy degree of freedom on the volume). Additional mappings can be supported where they make sense, please email sierra-help@sandia.gov if you have such a use case.

If either no valid mapping or multiple valid mappings are found the simulation will terminate during initialization.

31.2 Bulk Fluid Element

Scope: Equation System

```
Begin Bulk Fluid Element Name

    Bulk Coordinates {=|are|is} Parameters...

    Bulk Element Pressure {=|are|is} VolType [Parameters]...

    Bulk Element Volume {=|are|is} VolType [Parameters]...

    Bulk Eq EquationName For AssociatedDoF [ {of|species} SpeciesName |{in|material_phase}
    MaterialPhaseName ]Using ElementType With Terms...

    Bulk Source For EquationName {=|are|is} SrcType [Parameters]...

    Calculate Volume From Enclosing Surface

    Initial Pressure {=|are|is} p

    Initial Temperature {=|are|is} t

    Material {=|are|is} MatName

    Reinitialize Dof Name {=|are|is} value

    Temperature Is Celsius

    Update Bulk Volume Pressure [ Atmospheric {=|are|is} Value ]

    Use Block BlockName

End
```

Summary — Defines a bulk fluid element that can be used by a convective flux boundary condition.

Description — Generally speaking, the reference temperature used in Newton's Law of Cooling and is a known quantity because the fluid with which it is associated is modeled as an infinite reservoir. However, if the size of this reservoir is finite, then its temperature can be affected by the energy transfer across the surface in question. This situation can be modeled by the bulk fluid element, wherein the energy of the reservoir is determined by a finite volume conservation equation.

For a bulk fluid element with a name specified as *name*, the thermodynamic properties such as density, volume and temperature may be accessed via global variables. Density is accessed using *name*_RHO, volume by *name*_V, temperature at the current state by *name*_T and temperature at the old state by *name*_TOld.

31.2.1 Bulk Coordinates

Scope: Bulk Fluid Element

`Bulk Coordinates {=|are|is}` *`Parameters...`*

Parameter	Value	Default
Parameters	`real...`	`undefined`

Summary Specifies the physical coordinates for the bulk node.

31.2.2 Bulk Element Pressure

Scope: Bulk Fluid Element

`Bulk Element Pressure {=|are|is}` *`VolType [Parameters]...`*

Parameter	Value	Default
VolType	`string`	`undefined`
Parameters	`[string]...`	`undefined`

Summary Specifies the pressure of the bulk element with type Pressure Type (CONSTANT, USER) for bulk element defined by the specifications of the parameter list.

This command should not be used when the bulk element is associated with a pressurization zone model since initial pressure will instead be associated with the pressurization zone as a whole.

31.2.3 Bulk Element Volume

Scope: Bulk Fluid Element

`Bulk Element Volume {=|are|is}` *`VolType [Parameters]...`*

Parameter	Value	Default
VolType	`string`	`undefined`
Parameters	`[string]...`	`undefined`

Summary Specifies the volume of the bulk element with type VolType (CONSTANT, USER) for bulk element defined by the specifications of the parameter list.

For problems in which the mesh is deforming one should consider using CALCULATE VOLUME FROM ENCLOSING SURFACE instead.

31.2.4 Bulk Eq

Scope: Bulk Fluid Element

`Bulk Eq` *`EquationName`* `For` *`AssociatedDoF`* `[ {of|species}` *`SpeciesName`* `|{in|material_phase}` *`MaterialPhaseName`* `]Using` *`ElementType`* `With` *`Terms...`*

Parameter	Value	Default
EquationName	`string`	`undefined`
AssociatedDoF	`string`	`undefined`
SpeciesName	`string`	`undefined`
MaterialPhaseName	`string`	`undefined`
ElementType	`string`	`undefined`
Terms	`string...`	`undefined`

Summary Specifies an equation to solve on the bulk node. User provides the degree of freedom associated with the equation, the element type, and the terms in the equation that are active.

31.2.5 Bulk Source For

Scope: Bulk Fluid Element

`Bulk Source For` *`EquationName`* `{=|are|is}` *`SrcType [Parameters]...`*

Parameter	Value	Default
EquationName	`string`	`undefined`
SrcType	`string`	`undefined`
Parameters	`[string]...`	`undefined`

Summary Indicates a source term of model type (i.e. CONSTANT, USER, CALORE_USER_SUB) and associated model Parameters for the given EquationName on the bulk element.

31.2.6 Calculate Volume From Enclosing Surface

Scope: Bulk Fluid Element

Summary Specifies that the volume of the bulk element is to be calculated at each timestep from an enclosing surface. The surface must be closed (although Aria currently does not currently check for this) and is defined by the extent of the flux boundary condition which uses this bulk element.

31.2.7 Initial Pressure

Scope: Bulk Fluid Element

`Initial Pressure {=|are|is}` *`p`*

Parameter	Value	Default
p	`real`	`0.0`

Summary Specifies the initial pressure of the bulk node.

This command should not be used when the bulk element is associated with a pressurization zone model since initial pressure will instead be associated with the pressurization zone as a whole.

31.2.8 Initial Temperature

Scope: Bulk Fluid Element

`Initial Temperature {=|are|is}` *t*

Parameter	Value	Default
t	`real`	`REAL_MAX`

Summary Specifies the initial temperature of the bulk node.

31.2.9 Material

Scope: Bulk Fluid Element

`Material {=|are|is}` *MatName*

Parameter	Value	Default
MatName	`string`	`undefined`

Summary Specifies that the material properties of the bulk element are defined in the material block named matName.

31.2.10 Reinitialize Dof

Scope: Bulk Fluid Element

`Reinitialize Dof` *Name* `{=|are|is}` *value*

Parameter	Value	Default
Name	`string`	`undefined`
value	`real`	`undefined`

Summary Requests reinitialization of a bulk node DOF. This will allow override of the file supplied bulk node DOF value when using an initial condition from file. Here the initial value supplied from file will be replaced by a specified value.

31.2.11 Temperature Is Celsius

Scope: Bulk Fluid Element

Summary Specifies the initial temperature of the bulk node is given in degrees Celsius. Internal to the code this is relevant only when performing updates of the bulk element pressure via an ideal gas law which requires an absolute temperature.

31.2.12 Update Bulk Volume Pressure

Scope: Bulk Fluid Element

Summary If a bulk volume is filled with an ideal gas then the pressure can be calculated calculated at each timestep provided that the initial temperature and pressure are given. Here it is assumed that the associated volume is closed and that the volume has been defined as constant or is being computed. If the bulk volume is being computed then one must also include the CALCULATE VOLUME FROM ENCLOSING SURFACE command line.

This command should not be used when the bulk element is associated with a pressurization zone model since pressure is instead associated with the pressurization zone as a whole. The pressurization zone model also allows for gas models other than ideal gas.

31.2.13 Use Block

Scope: Bulk Fluid Element

`Use Block` *`BlockName`*

Parameter	Value	Default
BlockName	`string`	`undefined`

Summary Specifies that the given element block should be used for the bulk node. The caveat here is that there must be only one element, one node and they must all satisfy the proper mesh object topologies. This option is used only if an element block for the bulk element is already present in the mesh. If the bulk element does not exist on the input mesh, the element block will be created and will exist on the output mesh.

31.3 Closed Surface Volume

Scope: Equation System

```
Begin Closed Surface Volume ModelName
    Add Surface SurfaceList...
    Utility Group {= | are | is} Frequency
End
```

Summary This command block defines the parameters for computation of the volume of a closed surface. The computed volume is available as a global variable with name *modelName*.

Description Defines a means for computing the volume of a closed surface and associate it with name *modelName*.

31.3.1 Add Surface

Scope: Closed Surface Volume

`Add Surface` *`SurfaceList...`*

Parameter	Value	Default
SurfaceList	`string...`	`undefined`

Summary Adds surfaces by name, (surface_*id*), to a mesh extent list.

Description This line command is used to add surfaces to a mesh extent list. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

31.3.2 Utility Group

Scope: Closed Surface Volume

`Utility Group {= | are | is}` *`Frequency`*

Parameter	Value	Default
Frequency	{begin_nonlinear_solve | end_nonlinear_solve | manually_run | post_accept_solution | post_iterate | post_linear_solve | post_nonlinear_solve | pre_iterate | pre_linear_solve | pre_nonlinear_solve | run_initially | run_once | run_post_linear_system_initialization | run_post_mesh_mod}	undefined

---

Summary Sets the point in the execution to evaluate utility

31.4 Restarting With Bulk Volume Element

The bulk volume element capability is supported for restarted simulations 3.10. It is worth noting that while bulk volume element is generally not part of the original mesh discretization the element is being created internal to the code and is also being added to the results file as well as the restart file if defined. In some cases one may wish to initialize a simulation from a previous restart file that does not contain the bulk volume element. In this case provision must be made to ignore the fact that the initialization does not include the bulk volume element so that the bulk volume element can be created anew.

Input for restart jobs subsequent to the bulk volume element being added to a model *should not* contain the BULK NODES IGNORE RESTART command line.

31.4.1 Bulk Nodes Ignore Restart

Scope:

Summary Causes bulk nodes to ignore the restart file.

31.5 Bulk Node Coupling

Scope: Equation System

`Begin Bulk Node Coupling` *`Name`*

```
    Additional Parameter PropertyName {=|are|is} ModelName [Parameters]...
    Bulk Nodes {=|are|is} Names1 Names2...
    Couple DofName Model {=|are|is} ModelName [Parameters]...
End
```

Summary Defines a coupling between equations on two bulk nodes.

Description This block command allows a user to couple equations on different bulk nodes to one another. For example, the density of gas on two bulk nodes can be coupled using a flow coefficient based on the relative pressure of the two bulk nodes. The same degree of freedom must be present on both bulk nodes, and both have to be in the same equation system in order for the coupling to be possible.

31.5.1 Additional Parameter

Scope: Bulk Node Coupling

`Additional Parameter` *PropertyName* `{=|are|is}` *ModelName* *[Parameters]...*

Parameter	Value	Default
PropertyName	string	undefined
ModelName	string	undefined
Parameters	[string]...	undefined

Summary Declare an additional parameter for the coupling models present. For example, the "K_FACTOR_FLOW" model for density or species coupling requires that a "K_FACTOR" be provided using this line command.

31.5.2 Bulk Nodes

Scope: Bulk Node Coupling

`Bulk Nodes {=|are|is}` $Names_1$ $Names_2$`...`

Parameter	Value	Default
Names	string_1 string_2...	undefined

Summary Specify the name of the bulk nodes involved in the coupling.

31.5.3 Couple

Scope: Bulk Node Coupling

`Couple` *DofName* `Model {=|are|is}` *ModelName* *[Parameters]...*

Parameter	Value	Default
DofName	string	undefined
ModelName	string	undefined
Parameters	[string]...	undefined

Summary Specify what degree of freedom to couple between the bulk nodes, and what model to use for the coupling.

Chapter 32

Toggle Reference

32.1 Feature Toggling

For transient analysis Aria provides a mechanism for enabling and disabling certain aspects of a model like boundary conditions, contact conditions, source contributions and element block physics during selected time periods. This option is useful when modeling a time sequence of discontinuous events and the capability will be subsequently referred to as "toggling". Toggling as applied to simple electric switches means that one either opens or closes the switch (i.e. the switch is on or off). Similarly, from an analysis perspective one simply defines a specific model feature to be either active or inactive. Toggling **does not** imply the ability to swap in another physics model or to apply different model parameters.

Generally speaking the use of toggling is justified when a change in model physics will result in a change to the structure of the system matrix or to the structure of linear system contributions. For example, change of a flux boundary condition to a Dirichlet boundary condition requires a change to the structure of the matrix contribution since unknown values are now specified and need not be computed. Similarly, deactivation of a element block will remove degrees-of-freedom (rows and columns) from the Jacobian matrix thus reducing the number of system unknowns. Note that a change of flux boundary condition input parameters (e.g. the convective coefficient) will modify the system matrix coefficients but not affect the structure of the associated linear system matrix contributions so feature toggling is not justified in this case. Instead parameter changes in time should be accounted for through the parameter model definition.

In order to invoke this option one needs to consider its possible effect on results. Toggling of a model feature makes its usage take on a different character than expected for standard use. In particular, invocation of toggling for a simulation feature requires that the analyst prescribe the active/inactive state of the feature for the time periods of interest. Syntactically speaking the definition of feature toggling consists of two parts, the toggle block and selection of its usage.

The active/inactive state history of a feature is defined using a TOGGLE BLOCK command block. Within the Aria input file the TOGGLE BLOCK command block appears outside of the application scope, i.e. outside of the Procedure definition. Here the time period aspect of this specification is handled by mapping the desired state to a time block of the simulation. On the other hand specific usage of the toggle feature will appear within the Region scope.

For boundary conditions, contact conditions and source contributions the association between a particular model feature is prescribed by referencing the named TOGGLE BLOCK in the definition of a model feature. For native Aria model feature toggling the association with a named TOGGLE BLOCK is made by supplying an argument *toggle* = *toggle_block_name* to the feature (IC, BC or Source term) command line. To define toggling with Calore style input command blocks 33.1 the association between a model feature is prescribed by including a USE TOGGLE BLOCK command line within that command block.

For element block toggling (toggling of all element block physics), the association between the block and a named TOGGLE BLOCK is established via a standalone USE TOGGLE BLOCK command line within the Region scope. For toggling of the element block associated with a bulk fluid element 31.1 we note that

the corresponding element block name is BLOCK_FOR_*bulk_element_name*.

Toggling is supported for flux boundary conditions, contact conditions and source terms by simply zeroing out their contributions. Thus the number of DOF in the linear system will not change. Toggling of Dirichlet boundary conditions and element blocks is currently supported by removing the element DOF from the linear system.

Oftentimes one may wish to toggle off a surface boundary condition and toggle on a variant of the same boundary condition. As an example, one might replace a convective flux boundary condition with constant reference temperature with one having a reference temperature defined by a tabular function. Since the original boundary condition still exists (but it's toggled off) it will conflict with the new boundary condition. Cases such as this will require that the second boundary condition be applied to a different surface (differently named) that overlies the surface used in the original boundary condition in order to resolve this conflict.

Toggling has been proven to be unsupported for material properties or enclosure definitions.

Usage of toggling is demonstrated in the outline below where we note that the time period within the TOGGLE BLOCK corresponds to one of the time periods defined within the Solution Control 14 command block.

```
Begin Sierra myJob
  .
  Begin Toggle Block first_toggle
   period = a_time_period_2
   state = active
  End
  .
  Begin Toggle Block second_toggle
   period = a_time_period_3
   state = inactive
  End
  .
  Begin Procedure My_Aria_Procedure
     .
    Begin Solution Control Description
      Use system main
      Begin system main
        Simulation start time = 0.0
        Simulation termination time =  3600.0
        Begin transient a_time_period_1
          advance My_Region
        End
        Begin transient a_time_period_2
          advance My_Region
        End
        Begin transient a_time_period_3
          advance My_Region
        End
      End

      Begin parameters for transient a_time_period_2
        start time = 0.0
        termination time = 2.0
        Begin parameters for Aria region My_Region
```

```
        .
        End parameters for Aria region My_Region
      End parameters for transient a_time_period_2

      Begin parameters for transient a_time_period_2
        start time = 2.0
        termination time = 5.0
        Begin parameters for Aria region My_Region
         .
        End parameters for Aria region My_Region
      End parameters for transient a_time_period_2

    End
    .
    Begin Aria Region My_Region
     .
     Use toggle block first_toggle for block_8
     .
     Begin command block ablock
       .
       Use toggle block second_toggle
     End
     .
    End
    .
  End
  .
End Sierra myJob
```

Note that if one wishes to replace a boundary condition (the physics model) then one can do so by first adding a duplicate surface mesh entity and adding a new toggle block corresponding to usage of the new physics model.

It is important to note that within a TOGGLE BLOCK only one event (active or inactive) can be specified for the selected time period(s). Time periods other than the ones specified within the TOGGLE BLOCK command block will assume the opposite character (active/inactive). With reference to the previous outline this behavior is demonstrated by simple examples.

For the following toggle block

```
Begin Toggle Block my_toggle
 period = a_time_period_2
 state = active
End
```

a feature is active for `a_time_period_2` and inactive for `a_time_period_1` and `a_time_period_3`. Similarly the same behavior could be obtained using

```
Begin Toggle Block my_toggle
 period = a_time_period_1 a_time_period_3
 state = inactive
End
```

32.2 Toggle Block

Scope: Sierra

```
Begin Toggle Block ModelName

    Freeze Element Block Solution State

    Period {= | are | is} Period_list...

    State {= | are | is} ToggleState

End
```

Summary — This command block defines the period(s), state and initial conditions for model feature toggling.

Description — Defines a generic entity toggling specification and associate it with name *modelName*. The same *modelName* can be used for various model features such as element blocks, boundary conditions, contact and volumetric sources.

32.2.1 Freeze Element Block Solution State

Scope: Toggle Block

Summary — Specify that the solution state remain the same when an element block has been toggled on.

Description — Specifies that the initial condition is not reset on any associated element block that has been toggled. The default behavior is to have the initial conditions reset on element blocks that have been toggled back on.

32.2.2 Period

Scope: Toggle Block

`Period {= | are | is} Period_list...`

Parameter	Value	Default
Period_list	`string...`	`undefined`

Summary — Specify the periods over which this toggling model is to be used.

Description — Defines the time periods over which this toggling model is to be used. Note that more than one PERIOD can be appear in the command block.

32.2.3 State

Scope: Toggle Block

`State {= | are | is}` *`ToggleState`*

Parameter	Value	Default
ToggleState	`{active	inactive}`	`undefined`

Summary — Specify the state for this toggling model over the periods it is to be used.

Description — Defines the toggle state for the time periods over which this toggling model is to be used. Note that more than one STATE can be appear in the command block.

32.2.4 Use Toggle Block

Scope:

`Use Toggle Block` *`ToggleName`* `[ {@ | at | for | in | on | over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary — Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

Chapter 33

Thermal Analysis Reference

33.1 General Thermal Analysis

Aria supports the solution of the energy equation for temperature as described an earlier section 5.3. following the paradigm of native Aria input. Here a user can also utilize any of the standard Aria input commands previously described in the initial conditions Chapter 8, boundary conditions Chapter 9 and volumetric source terms Chapter 11.

Previous users of the Calore code will find that many of the general input file commands for initial conditions, boundary conditions and volume sources are in fact recognized by Aria. However, there exist major differences between the two code inputs with regard to the material property definitions. Users transitioning from the Calore to Aria should review the chapter on Aria Material properties 4. While most of the property specifications differ only slightly, the most striking difference between the two command blocks is that the Aria material input requires an additional command line definition of a heat conduction model 4.15.

The Calore and Aria code code inputs also differ slightly in the nonlinear solution strategy. Generally speaking, a solution step in Aria cannot be advanced until the linear system is fully converged whereas in Calore the solution need not be converged for the step to advance. Calore and Aria also differ in the method of residual norm evaluation. Here Calore uses a residual norm scaled by the DOF whereas Aria uses an unscaled residual norm. Provisions are made in Aria to mimic the Calore code in the aforementioned behavior by use of additional input instructions, `ACCEPT SOLUTION AFTER MAXIMUM NONLINEAR ITERATIONS` and `USE DOF AVERAGED NONLINEAR RESIDUAL` described in a section on nonlinear solution specification 17.3.

Aria also provides support of user subroutines for initial condition, volumetric source, boundary condition or material property. When a user subroutine is named within a standard thermal analysis command block it must be referred to as a Calore_User_Sub. Usage of input file data within these user subroutines is facilitated by use of named Data Blocks that can be made available to the user subroutines. These Data Blocks fill the role formerly occupied by Real and Int command lines in the Calore.

As previously mentioned, one may setup the solution of a thermal problem using either Calore style input commands or native Aria input commands. It is worth noting that when using the native Aria commands for heat flux, the sign convention for heat flux in the boundary condition commands for constant flux 9.2.1 and user_function flux 9.2.35 differs from that in the Calore style of heat flux in that they are of opposite sign.

Aria supports a number of non-native (Calore style) command line blocks to facilitate definition of the thermal problem to be solved. These command blocks are defined in the sections which follow. Here input specification on portions of the FEM mesh can be simplified by aggregation of the parts into Mesh Groups 6.26.

33.2 Checking Conservation

Aria has a builtin energy conservation check that can be activated for the energy equation. This is accomplished by adding the following line to whatever block you specify the energy equation in (either an Equation System block or the Region block):

```
Check Conservation of Energy
```

When this is active, after each time step the accumulated internal energy is compared with the energy added from all boundaries and sources and the level of conservation is printed in the log file.

This is a conservation check targeted towards conduction problems. The balance calculations do not currently include advection or divergence sources and will not be accurate for compressible fluids or domains with advection at the boundaries.

33.3 Non-physical Temperature Solutions

The conservation of energy equation in its continuous form (and in the absence of source terms) satisfies a maximum principle. That is, for a steady-state problem the maximum and minimum temperatures in the domain must occur on the boundary of the domain, and for a transient problem on either the boundary or the initial condition. However, the discretized form of the energy equation does not necessarily satisfy a discrete version of the maximum principle (DMP). In particular for the standard Galerkin finite element method used in Aria there are conditions on the mesh quality that must be met for the diffusion operator to satisfy a DMP. For linear tetrahedral elements the condition is that there are no obtuse dihedral angles between faces of elements in the mesh [37]. Anisotropic thermal conductivity can also contribute to this effect [38]. Additionally, the consistent mass term can also lead to DMP violations in transient problems [39]. In practice this can lead to non-physical temperatures appearing in the solutions generated by Aria, most commonly when large heat fluxes are applied to an initially cold domain with a low thermal conductivity.

If non-physical temperature solutions are observed Aria supports an option to apply a nonlinear flux correction based on the work of Kuzmin et al. that restores a DMP on arbitrary meshes with arbitrary material properties [39, 40]. This option may be activated by using the "APPLY FLUX LIMITER STABILIZATION" line command, 41.1.1. For transient problems it is also essential to use the LUMPED_MASS form of the time derivative term. At present, this option is only intended for use with linear elements, not second order 10 node tetrahedral elements or 27 node hexahedral elements. Source terms also do not have any limiters applied to them at present, as a result they may still cause non-physical temperatures in some cases.

Additionally, applying the stabilization has several downsides so that we only recommend enabling it if problematic non-physical temperatures are observed without it. Solutions with the stabilization activated are more diffusive and have higher error than unstabilized solutions (though they do converge at the same order with mesh refinement). The stabilization operator is also nonlinear and can adversely affect the convergence of the Newton-Raphson iteration within each time step. Combined with the cost of calculating the stabilization terms this can have noticeable impact on simulation runtime.

33.4 Initial Condition

Scope: Aria Region

```
Begin Initial Condition BlockName
```

```
    Add Surface PartList...
    Add Volume PartList...
    All Volumes
    Node Subroutine {=|are|is} Name
    Temperature {=|are|is} t0
    Use Data Block Name
    Use File Variable File_variable_name For Aria_variable_name [ {of|species} SpeciesName
    |{in|material_phase} MaterialPhaseName ][ At Time Value ]
End
```

Summary Allows the specification of a user-defined initial conditions on any combination of volumes, surfaces and nodes.

Description There are three methods for setting the initial conditions of the thermal model. The temperature specification must be of exactly one type: e.g., constant, defined in a user subroutine, or read from a solution database file. If a user subroutine is used, real and integer data may be declared that will be local in scope to this instance of the boundary condition. This data may be accessed using a user query function.

33.4.1 Add Surface

Scope: Initial Condition

`Add Surface` *PartList...*

Parameter	Value	Default
PartList	`string...`	`undefined`

Summary Surface name to which this initial condition applies. May specify a space-delimited list.

33.4.2 Add Volume

Scope: Initial Condition

`Add Volume` *PartList...*

Parameter	Value	Default
PartList	`string...`	`undefined`

Summary Block names to which this initial condition applies. May specify a space-delimited list.

33.4.3 All Volumes

Scope: Initial Condition

Summary Apply this command block to all volumes in the mesh file.

33.4.4 Node Subroutine

Scope: Initial Condition

`Node Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies that the named user-defined subroutine be used. At most one subroutine name may be specified for a given quantity. This subroutine must conform to the "node signature" argument list type.

33.4.5 Temperature

Scope: Initial Condition

`Temperature {=|are|is}` *`t0`*

Parameter	Value	Default
t0	`real`	`undefined`

Summary Set the initial temperature to a constant value.

33.4.6 Use Data Block

Scope: Initial Condition

`Use Data Block` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

33.4.7 Use File Variable

Scope: Initial Condition

`Use File Variable` *`File_variable_name`* `For` *`Aria_variable_name`* `[ {of|species}` *`SpeciesName`* `|{in|material_phase}` *`MaterialPhaseName`* `][ At Time` *`Value`* `]`

Parameter	Value	Default
File_variable_name	`string`	`undefined`
Aria_variable_name	`string`	`undefined`
SpeciesName	`string`	`undefined`
MaterialPhaseName	`string`	`undefined`

Summary Assign file nodal variable to valid registered nodal variable for initial condition.

Description This line command specifies that the named Aria variable be initialized from a variable contained in the mesh file that is assigned to the region. For example, this line command may be used to initialize a transient calculation from a previously performed steady-state calculation in the following way. Perform the steady-state calculation and output the temperature. Then copy that output file and use it as a mesh file for the transient calculation. Finally, use this line command to tie the variable named "file_variable_name" in the mesh file to the aria variable named "aria_variable_name". Here "aria_variable_name" must be set to nonlinear_solution->temperature.

The closest time step to the time value found the IO database will be used for the assignment. For an initial condition taken from a steady state results file, omit the optional time command. The optional command is necessary if the initial condition is coming from a file with multiple timesteps.

NOTES: 1) Current framework services do not allow restricting this initialization by mesh subset. e.g., if you want to initialize the temperature, then all nodes in the mesh will be initialized. 2) Today, Aria only supports initialization of temperature field. This restriction may be removed in the future so that chemistry variables or user-defined variables may also be initialized.

33.5 Temperature Boundary Condition

Scope: Equation System

```
Begin Temperature Boundary Condition BC name
    Add Surface SurfaceList...
    Field {=|are|is} VariableName
    Integer Data Values...
    Node Subroutine {=|are|is} Name
    Real Data Values...
    Temperature {=|are|is} Value
    Temperature Fortran Subroutine {=|are|is} Name
    Temperature Node Variable {=|are|is} Name
    Temperature Scale Factor {=|are|is} Magnitude
    Temperature Time Function {=|are|is} FunctionName
    Use Data Block Name
    Use Death Name
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList...  ]
    Value {=|are|is} VariableValue
End
```

Summary Specified temperature boundary condition.

Description For this boundary condition, the conservation equations are discarded, and the temperature is completely determined by this boundary condition. The temperature specification must be of exactly one type: e.g., constant, time-dependent, or defined in a user subroutine. If a user subroutine is used, real and integer data may be declared that will be local in scope to this instance of the boundary condition. This data may be accessed using a user query function.

33.5.1 Add Surface

Scope: Temperature Boundary Condition

`Add Surface` *`SurfaceList...`*

Parameter	Value	Default
SurfaceList	`string...`	`undefined`

Summary — Adds surfaces, by name, to a boundary condition's extent.

Description — This line command is used to add surfaces to the extent of a boundary condition. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.5.2 Field

Scope: Temperature Boundary Condition

`Field {= | are | is}` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary — This command can be used with Dirichlet conditions for any solution unknown used in Aria. The FIELD command argument defines the name of the variable to be specified with this boundary condition.

33.5.3 Integer Data

Scope: Temperature Boundary Condition

`Integer Data` *`Values...`*

Parameter	Value	Default
Values	`integer...`	`undefined`

Summary — List of integer data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.5.4 Node Subroutine

Scope: Temperature Boundary Condition

`Node Subroutine {= | are | is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies that the named user-defined subroutine be used. At most one subroutine name may be specified for a given quantity. This subroutine must conform to the "node signature" argument list type.

33.5.5 Real Data

Scope: Temperature Boundary Condition

`Real Data` *`Values...`*

Parameter	Value	Default
Values	`real...`	`undefined`

Summary List of real data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.5.6 Temperature

Scope: Temperature Boundary Condition

`Temperature {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify the constant value of the temperature. Using this command is equivalent to using both the FIELD = temperature and VALUE = real_value command lines.

33.5.7 Temperature Fortran Subroutine

Scope: Temperature Boundary Condition

`Temperature Fortran Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a FORTRAN user subroutine that will be used to calculate the nodal temperature boundary condition.

33.5.8 Temperature Node Variable

Scope: Temperature Boundary Condition

`Temperature Node Variable {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of the node variable to use for the temperature associated with this boundary condition.

Description The indicated node variable must be a legal Aria variable. This variable is defined from the input file in the USER VARIABLE command block or with the USER FIELD command. Two distinct use cases may arise, 1) the variable is calculated in another SIERRA region, e.g., Fuego, and transferred to Aria 2) the variable exists in the input grid file and is populated using the READ VARIABLE command from within the USER FIELD command block.

33.5.9 Temperature Scale Factor

Scope: Temperature Boundary Condition

`Temperature Scale Factor {=|are|is}` *`Magnitude`*

Parameter	Value	Default
Magnitude	`real`	`undefined`

Summary Specifies the magnitude by which the distribution factors will be multiplied in order to determine the boundary condition. If this line command is used, node distribution factors must be defined in the input mesh for all nodesets which are associated with this boundary condition.

Description If this line command is used, node distribution factors must be defined in the input mesh for all nodesets which are associated with this boundary condition. There is currently an inconsistency with respect to temperature boundary conditions and distribution factors: It is not legal to specify sideset distribution factors with the temperature boundary condition: nodesets must be used. Distribution factors also do not work with h-adaptivity, as their values are not currently interpolated to the new nodes. In any case, for temperature boundary conditions, in order for this interpolation to occur, then a sideset would have to be used, which is not supported at this time.

33.5.10 Temperature Time Function

Scope: Temperature Boundary Condition

`Temperature Time Function {=|are|is}` *`FunctionName`*

Parameter	Value	Default
FunctionName	`string`	`undefined`

Summary Name of the time-dependent function that specifies the temperature.

33.5.11 Use Data Block

Scope: Temperature Boundary Condition

`Use Data Block` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

33.5.12 Use Death

Scope: Temperature Boundary Condition

`Use Death` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Associates this boundary condition with element faces that are created as a result of element death due to the specified element death command block. Note that this line command makes the owning boundary condition have a dynamic extent that changes as elements die.

Description This line command specifies that the faces that are created as a result of the element death criterion in the named element death command block are added to the extent of this boundary condition. A boundary condition may specify an extent via the "add surface" line command, or the "use death" line command. In this way it is possible to have a boundary condition that is initially empty and then grows dynamically with the surface created as a result of the element death. At least one of these methods must be used.

A boundary condition may be associated with more than one death command block by including this line command more than once. However, to avoid issues of precedence, there is a rule that a given element may not die for more than one reason.

33.5.13 Use Toggle Block

Scope: Temperature Boundary Condition

`Use Toggle Block` *`ToggleName`* `[ {@ | at | for | in | on | over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.5.14 Value

Scope: Temperature Boundary Condition

`Value {= | are | is}` *`VariableValue`*

Parameter	Value	Default
VariableValue	`string`	`undefined`

Summary This command is used in conjunction with the FIELD command. A constant value of the variable given in the FIELD command is specified with this boundary condition.

33.6 Heat Flux Boundary Condition

Scope: Equation System

```
Begin Heat Flux Boundary Condition Name

    Add Surface SurfaceList...

    Element Subroutine {=|are|is} Name

    Equation {=|are|is} EquationName

    Field Scaling {=|are|is} Scale_factor

    Flux {=|are|is} Value

    Flux Encore Function {=|are|is} Name

    Flux Fortran Subroutine {=|are|is} Name

    Flux Node Variable {=|are|is} Name [ Multiplier {=|are|is} Value ]

    Flux Scale Factor {=|are|is} Value

    Flux Temperature Function {=|are|is} Namep

    Flux Time Function {=|are|is} Name

    Flux Vector Node Variable {=|are|is} Name [ Multiplier {=|are|is} Value ]

    Ignore Flux Coverage

    Influx Vector Node Variable {=|are|is} Name [ Multiplier {=|are|is} Value ]

    Integer Data Values...

    Integrated Flux Output VariableName

    Integrated Power Output VariableName

    Real Data Values...

    Scaling Time Function {=|are|is} Function

    Scaling With Global Variable {=|are|is} name

    Use Data Block Name

    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList...  ]

End
```

Summary — This command block specifies that a known heat flux is to be applied normal to the given surface.

Description — The flux can vary in arbitrary way, e.g. it may be a specified constant, a function of temperature, time etc. You may only specify one kind of flux e.g it is not legal to specify a constant value and a time dependent function. If you want to add two such fluxes on a single surface, then either write a user subroutine or use two different command blocks. If a function name is given then the function must be defined in a function definition command block.

33.6.1 Add Surface

Scope: Heat Flux Boundary Condition

`Add Surface` *`SurfaceList...`*

Parameter	Value	Default
SurfaceList	`string...`	`undefined`

Summary — Adds surfaces, by name, to a boundary condition's extent.

Description — This line command is used to add surfaces to the extent of a boundary condition. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.6.2 Element Subroutine

Scope: Heat Flux Boundary Condition

`Element Subroutine {= | are | is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Specifies that the named user-defined subroutine be used. At most one subroutine name may be specified for a given quantity. This subroutine must conform to the "element signature" argument list type.

33.6.3 Equation

Scope: Heat Flux Boundary Condition

`Equation {= | are | is}` *`EquationName`*

Parameter	Value	Default
EquationName	`string`	`undefined`

Summary — This command can be used to apply the Heat Flux conditions for an equation other than the ENERGY equation.

33.6.4 Field Scaling

Scope: Heat Flux Boundary Condition

`Field Scaling {= | are | is}` *`Scale_factor`*

Parameter	Value	Default
Scale_factor	`real`	`undefined`

Summary This command is only relevant when the FLUX NODE VARIABLE, FLUX VECTOR NODE VARIABLE or INFLUX VECTOR NODE VARIABLE command line option is present. The command enables scaling of the Field by a single value for the surfaces defined within the scope of the current Heat Flux command block. For vector Fields all components will be scaled by the provided value.

33.6.5 Flux

Scope: Heat Flux Boundary Condition

`Flux {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary This line command specifies a constant value of heat flux.

33.6.6 Flux Encore Function

Scope: Heat Flux Boundary Condition

`Flux Encore Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of an Encore function that will be used to calculate the flux boundary condition. Note that the sign convention for flux into the domain requires that the Encore function be negatively signed.

33.6.7 Flux Fortran Subroutine

Scope: Heat Flux Boundary Condition

`Flux Fortran Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a FORTRAN user subroutine that will be used to calculate the flux boundary condition.

33.6.8 Flux Node Variable

Scope: Heat Flux Boundary Condition

`Flux Node Variable {=|are|is}` *`Name`* `[` *`Multiplier`* `{=|are|is}` *`Value`* `]`

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of the node variable to use for the flux that is associated with this boundary condition. Here the node variable is assumed to be the product of the surface unit normal vector with the nodal heat flux. Optionally, the value can be scaled by product of Multiplier value and FIELD SCALING.

Description The indicated node variable must be a legal Aria variable. In most cases this variable is defined by the user in the user variable definition command block. The node variable is interpolated to the quadrature points during the evaluation of flux term contributions. If need be, both the sign and magnitude of the node variable can be modified by using the FIELD SCALING command line. Typically, the node variable would be calculated in another SIERRA region, e.g., Fuego, and transferred to Aria.

33.6.9 Flux Scale Factor

Scope: Heat Flux Boundary Condition

`Flux Scale Factor {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies the magnitude by which the distribution factors will be multiplied in order to determine the boundary condition. If this line command is used, sideset distribution factors must be defined in the input mesh for all surfaces which are associated with this boundary condition.

Description The flux on each face is computed by interpolating the distribution factors to the Gauss points and then multiplying the result by the given magnitude. Distribution factors do not currently work with h–adaptivity, since their values are not currently interpolated to the new nodes.

33.6.10 Flux Temperature Function

Scope: Heat Flux Boundary Condition

`Flux Temperature Function {=|are|is}` *`Namep`*

Parameter	Value	Default
Namep	`string`	`undefined`

Summary Specifies the name of a temperature dependent heat flux function that is used to calculate the normal flux.

33.6.11 Flux Time Function

Scope: Heat Flux Boundary Condition

`Flux Time Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a time dependent heat flux function that is used to calculate the normal flux.

33.6.12 Flux Vector Node Variable

Scope: Heat Flux Boundary Condition

`Flux Vector Node Variable {=|are|is}` *Name* `[` *Multiplier* `{=|are|is}` *Value* `]`

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of the vector node variable to use for the flux that is associated with this boundary condition. Here the flux contribution for a surface will be evaluated by forming a dot product of the unit normal vector with the vector node variable, without regard to the sign of the dot product. Optionally, the vector values can be scaled by product of Multiplier value and FIELD SCALING.

Description The indicated node variable must be a legal Aria vector variable. In most cases this variable is defined by the user in the user variable definition command block. The node variable is interpolated to the quadrature points during the evaluation of flux term contributions. If need be, both the sign and magnitude of the node variable can be modified by using the FIELD SCALING command line. Typically, the vector node variable would be calculated in another SIERRA region, e.g., Fuego, and transferred to Aria.

33.6.13 Ignore Flux Coverage

Scope: Heat Flux Boundary Condition

Summary This command causes the code to ignore the flux coverage Field when contact is present. Thus the flux will be applied even if the BC is being set on a contact surface.

33.6.14 Influx Vector Node Variable

Scope: Heat Flux Boundary Condition

`Influx Vector Node Variable {=|are|is}` *Name* `[` *Multiplier* `{=|are|is}` *Value* `]`

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of the node variable to use for the flux that is associated with this boundary condition. Here the applied flux will be formed by the dot product of the surface unit normal vector with the nodal variable which produce an influx contribution. Optionally, the value can be scaled by product of Multiplier value and FIELD SCALING.

Description The indicated node variable must be a legal Aria variable. In most cases this variable is defined by the user in the User Variable definition command block. The node variable is interpolated to the quadrature points during the evaluation of flux term contributions. If need be, both

the sign and magnitude of the node variable can be modified by using the FIELD SCALING command line. Typically, the node variable would be calculated in another SIERRA region, e.g., Fuego, and transferred to Aria.

33.6.15 Integer Data

Scope: Heat Flux Boundary Condition

`Integer Data` *`Values...`*

Parameter	Value	Default
Values	`integer...`	`undefined`

Summary — List of integer data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.6.16 Integrated Flux Output

Scope: Heat Flux Boundary Condition

`Integrated Flux Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary — Calculate the average flux associated with this flux boundary condition.

Description — This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power. This power is then divided by the total area of the surface to obtain the average flux on the surface, and stored in a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.6.17 Integrated Power Output

Scope: Heat Flux Boundary Condition

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary — Calculate the total power associated with this flux boundary condition.

Description — This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.6.18 Real Data

Scope: Heat Flux Boundary Condition

`Real Data` *`Values...`*

Parameter	Value	Default
Values	`real...`	`undefined`

Summary List of real data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.6.19 Scaling Time Function

Scope: Heat Flux Boundary Condition

`Scaling Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary This command is only relevant when the FLUX NODE VARIABLE, FLUX VECTOR NODE VARIABLE or INFLUX VECTOR NODE VARIABLE command line option is present. This command enables time function scaling of the field variable for the surfaces defined within the scope of the current Heat Flux command block.

33.6.20 Scaling With Global Variable

Scope: Heat Flux Boundary Condition

`Scaling With Global Variable {=|are|is}` *`name`*

Parameter	Value	Default
name	`string`	`undefined`

Summary This command is only relevant when the FLUX NODE VARIABLE, FLUX VECTOR NODE VARIABLE or INFLUX VECTOR NODE VARIABLE command line option is present. The command enables scaling of the Field by a global variable for the surfaces defined within the Heat Flux command block. For vector Fields all components will be scaled by the named scalar global variable. The scalar global variable itself must be defined at the Region scope as type Real with length 1.

33.6.21 Use Data Block

Scope: Heat Flux Boundary Condition

`Use Data Block` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

33.6.22 Use Toggle Block

Scope: Heat Flux Boundary Condition

`Use Toggle Block` *ToggleName* `[ {@|at|for|in|on|over}` *ElementBlockList...* `]`

Parameter	Value	Default
ToggleName	string	undefined

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.7 Laser Heat Flux Boundary Condition

Scope: Equation System

`Begin Laser Heat Flux Boundary Condition` *Name*

`Absorption Coefficient {=|are|is}` *value*

`Absorption Coefficient Data File {=|are|is}` *Filename*

`Add Surface` *SurfaceList...*

`Angular Velocity {=|are|is}` *value*

`Beam Diameter {=|are|is}` *laser_diameter*

`Beam Radius {=|are|is}` *value*

`Circular Path Center {=|are|is}` *Point...*

`Circular Path Start Vector {=|are|is}` *Value...*

`Compute Visibility Field {=|are|is}` *FieldName*

`Effective Beam Radius {=|are|is}` *value*

`Element Subroutine {=|are|is}` *Name*

`Flux {=|are|is}` *Value*

`Flux Fortran Subroutine {=|are|is}` *Name*

`Flux Type {=|are|is}` *LaserDistributionType*

`Integer Data` *Values...*

`Integrated Flux Output` *VariableName*

`Integrated Power Output` *VariableName*

`Max Degrees {=|are|is}` *value*

`Path Function {=|are|is}` *Function*

`Path Radius {=|are|is}` *value*

`Power {=|are|is}` *laser_power*

`Power Encore Function {=|are|is}` *Function*

```
    Power Time Function {=|are|is} Function
    Real Data Values...
    Rotation Axis Vector {=|are|is} Value...
    Source Direction Vector {=|are|is} Value...
    Source Start Location {=|are|is} Point...
    Source Velocity Vector {=|are|is} Value...
    Start Time {=|are|is} Value
    Stop Time {=|are|is} Value
    Use Data Block Name
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList... ]
End
```

Summary This command block specifies that a laser heat flux is to be applied on the given surface.

Description The laser heat flux depends upon the laser beam characteristics and will vary with time and space in arbitrary manner as specified by the path description and its position relative to the surface normal.

33.7.1 Absorption Coefficient

Scope: Laser Heat Flux Boundary Condition

```
Absorption Coefficient {=|are|is} value
```

Parameter	Value	Default
value	real	undefined

Summary Specifies the surface absorption coefficient.

33.7.2 Absorption Coefficient Data File

Scope: Laser Heat Flux Boundary Condition

```
Absorption Coefficient Data File {=|are|is} Filename
```

Parameter	Value	Default
Filename	string	undefined

Summary Specifies filename to read absorption coefficient data from.

33.7.3 Add Surface

Scope: Laser Heat Flux Boundary Condition

```
Add Surface SurfaceList...
```

Parameter	Value	Default
SurfaceList	string...	undefined

Summary Adds surfaces, by name, to a boundary condition's extent.

Description This line command is used to add surfaces to the extent of a boundary condition. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.7.4 Angular Velocity

Scope: Laser Heat Flux Boundary Condition

`Angular Velocity {=|are|is}` *value*

Parameter	Value	Default
value	`real`	`undefined`

Summary Specifies angular velocity of laser movement in revolutions per unit of time.

33.7.5 Beam Diameter

Scope: Laser Heat Flux Boundary Condition

`Beam Diameter {=|are|is}` *laser_diameter*

Parameter	Value	Default
laser_ diameter	`real`	`undefined`

Summary Specifies laser spot size.

33.7.6 Beam Radius

Scope: Laser Heat Flux Boundary Condition

`Beam Radius {=|are|is}` *value*

Parameter	Value	Default
value	`real`	`undefined`

Summary Specifies radius of laser beam.

33.7.7 Circular Path Center

Scope: Laser Heat Flux Boundary Condition

`Circular Path Center {=|are|is}` *Point...*

Parameter	Value	Default
Point	`real...`	`undefined`

Summary Specifies the center of circular path for laser source travel.

33.7.8 Circular Path Start Vector

Scope: Laser Heat Flux Boundary Condition

`Circular Path Start Vector {=|are|is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary Specifies the starting location on circular path, a radius vector relative to CIRCULAR PATH CENTER.

33.7.9 Compute Visibility Field

Scope: Laser Heat Flux Boundary Condition

`Compute Visibility Field {=|are|is}` *`FieldName`*

Parameter	Value	Default
FieldName	`string`	`undefined`

Summary Specifies visibility Field to be computed.

33.7.10 Effective Beam Radius

Scope: Laser Heat Flux Boundary Condition

`Effective Beam Radius {=|are|is}` *`value`*

Parameter	Value	Default
value	`real`	`undefined`

Summary Specifies effective radius of laser beam.

33.7.11 Element Subroutine

Scope: Laser Heat Flux Boundary Condition

`Element Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies that the named user-defined subroutine be used. At most one subroutine name may be specified for a given quantity. This subroutine must conform to the "element signature" argument list type.

33.7.12 Flux

Scope: Laser Heat Flux Boundary Condition

`Flux {=|are|is} Value`

Parameter	Value	Default
Value	`real`	`undefined`

Summary This line command specifies a constant value of heat flux.

33.7.13 Flux Fortran Subroutine

Scope: Laser Heat Flux Boundary Condition

`Flux Fortran Subroutine {=|are|is} Name`

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a FORTRAN user subroutine that will be used to calculate the flux boundary condition.

33.7.14 Flux Type

Scope: Laser Heat Flux Boundary Condition

Summary Selects the laser flux distribution model applied to a portion of the underlying surface as defined by a geometric path function. Flux types produce either a Gaussian flux with maximum intensity at the center of the laser spot or constant flux.

33.7.15 Integer Data

Scope: Laser Heat Flux Boundary Condition

`Integer Data Values...`

Parameter	Value	Default
Values	`integer...`	`undefined`

Summary List of integer data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.7.16 Integrated Flux Output

Scope: Laser Heat Flux Boundary Condition

`Integrated Flux Output VariableName`

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Calculate the average flux associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power. This power is then divided by the total area of the surface to obtain the average flux on the surface, and stored in a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.7.17 Integrated Power Output

Scope: Laser Heat Flux Boundary Condition

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Calculate the total power associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.7.18 Max Degrees

Scope: Laser Heat Flux Boundary Condition

`Max Degrees {=|are|is}` *`value`*

Parameter	Value	Default
value	`real`	`undefined`

Summary Specifies maximum degrees of laser rotational motion.

33.7.19 Path Function

Scope: Laser Heat Flux Boundary Condition

`Path Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies a time and position function for which a laser source is applied in the named volume.

33.7.20 Path Radius

Scope: Laser Heat Flux Boundary Condition

`Path Radius {=|are|is}` *`value`*

Parameter	Value	Default
value	`real`	`undefined`

Summary Specifies radius for circular laser path.

33.7.21 Power

Scope: Laser Heat Flux Boundary Condition

`Power {=|are|is}` *`laser_power`*

Parameter	Value	Default
laser_power	`real`	`undefined`

Summary Specifies strength of the laser source.

33.7.22 Power Encore Function

Scope: Laser Heat Flux Boundary Condition

`Power Encore Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies that power be applied with the named Encore function in the named volume.

33.7.23 Power Time Function

Scope: Laser Heat Flux Boundary Condition

`Power Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies that power be applied as a function of time in the named volume.

33.7.24 Real Data

Scope: Laser Heat Flux Boundary Condition

`Real Data` *`Values...`*

Parameter	Value	Default
Values	`real...`	`undefined`

Summary List of real data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.7.25 Rotation Axis Vector

Scope: Laser Heat Flux Boundary Condition

`Rotation Axis Vector {=|are|is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary Specifies the axis of rotation for circular source travel, relative to CIRCULAR PATH CENTER.

33.7.26 Source Direction Vector

Scope: Laser Heat Flux Boundary Condition

`Source Direction Vector {=|are|is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary Specifies the vector orientation of the laser source.

33.7.27 Source Start Location

Scope: Laser Heat Flux Boundary Condition

`Source Start Location {=|are|is}` *`Point...`*

Parameter	Value	Default
Point	`real...`	`undefined`

Summary Specifies the physical coordinate location of the laser source.

33.7.28 Source Velocity Vector

Scope: Laser Heat Flux Boundary Condition

`Source Velocity Vector {=|are|is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary Specifies the velocity vector of laser source motion.

33.7.29 Start Time

Scope: Laser Heat Flux Boundary Condition

`Start Time {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies the time at which the laser beam is activated.

33.7.30 Stop Time

Scope: Laser Heat Flux Boundary Condition

`Stop Time {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies the time at which the laser beam is deactivated.

33.7.31 Use Data Block

Scope: Laser Heat Flux Boundary Condition

`Use Data Block` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

33.7.32 Use Toggle Block

Scope: Laser Heat Flux Boundary Condition

`Use Toggle Block` *`ToggleName`* `[ {@|at|for|in|on|over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.8 Convection Heat Transfer

A broad class of thermal applications involve heat transfer between a solid volume and a surrounding fluid. In numerical simulations these interactions are modeled using various surface boundary conditions depending upon the detail to which the fluid flow is resolved. If both heat transfer in the solid model and fluid flow are explicitly modeled then it is often appropriate to solve the conjugate heat transfer problem. However,

in many cases the expense of this calculation may not be justified and one reverts to alternative methods of modeling these effects.

In many situations it may be appropriate to simply characterize a bulk behavior of the fluid and employ a convective flux boundary condition of the form

$$q = h(T)(T_s - T_b)$$

where q is the heat flux per unit surface area, $h(T)$ is the surface heat transfer coefficient or convection coefficient, T_s is the surface temperature and T_b is the bulk fluid temperature. Many of the engineering models based upon experimental data for given configurations include variations of the values that enter into the expression above. In particular the values of $h(T)$ and T_b are often linked to representative fluid flow conditions.

For quick estimates of heat transfer, textbooks provide a tabulated range of surface heat transfer coefficients for general conditions. Furthermore, heat transfer handbooks provide a collection of heat transfer correlations from which convective coefficients can be computed. These correlations include empirical constants and are functions dimensionless numbers which characterize the bulk fluid flow and the bulk fluid temperature.

The Sierra Thermal module includes a number of different ways for specifying $h(T)$ and T_b. Additionally module provides a catalog of heat transfer correlations 34.1.

33.9 Convective Flux Boundary Condition

Scope: Equation System

```
Begin Convective Flux Boundary Condition Name
    Add Surface SurfaceList...
    Average Temperature Variable {=|are|is} NName
    Convective Coefficient {=|are|is} Value
    Convective Coefficient Encore Function {=|are|is} Name
    Convective Coefficient Fortran Subroutine {=|are|is} Name
    Convective Coefficient Node Variable {=|are|is} Name
    Convective Coefficient Scale Factor {=|are|is} Magnitude
    Convective Coefficient Subroutine {=|are|is} Name
    Convective Coefficient Temperature Difference Function {=|are|is} NName
    Convective Coefficient Temperature Function {=|are|is} NName
    Convective Coefficient Time Function {=|are|is} Name
    Equation {=|are|is} EquationName
    Ignore Flux Coverage
    Integer Data Values...
    Integrated Flux Output VariableName
    Integrated Power Output VariableName
    Real Data Values...
    Ref Temp Convective Coefficient Subroutine {=|are|is} Name [ UsingRefTemp ]
    Reference Temperature {=|are|is} Value
```

```
    Reference Temperature Fortran Subroutine {=|are|is} Name
    Reference Temperature Global Variable {=|are|is} GlobalVariableName
    Reference Temperature Node Variable {=|are|is} Name
    Reference Temperature Subroutine {=|are|is} Name
    Reference Temperature Temperature Function {=|are|is} Name
    Reference Temperature Time Function {=|are|is} FunctionName
    Scaled Convective Coefficient Subroutine {=|are|is} Name FieldName {=|are|is} Field

    Scaled Ref Temp Convective Coefficient Subroutine {=|are|is} Name FieldName {=|
    are|is} Field
    Uq Flux Multiplier {=|are|is} Value
    Use Advective Bar Name [ BulkNodes ]
    Use Bulk Element Name
    Use Correlation Convection Model Name
    Use Data Block Name
    Use Enclosure Name
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList...  ]
    User Field Mask {=|are|is} Name [ Threshold {=|are|is} Value ]
    User Field Scaling {=|are|is} Name
End
```

Summary — This command block specifies heat transfer on a boundary surface that can be modeled using Newton's law of cooling.

Description — Newton's law of cooling specifies that the heat flux normal to a surface is proportional to the difference between the unknown temperature of the surface and some reference temperature of the fluid in which the surface is immersed: $q_n = h(T - T_r)$. The convection coefficient, h, and the reference temperature, T_r, may be specified in several ways as explained in detail below. In particular, note that the reference temperature is usually a known quantity, because the fluid with which it is associated is modeled as an infinite reservoir. However, if the size of this reservoir is finite, then its temperature can be affected by the energy transfer across the surface in question. This situation can be modeled by the bulk fluid element, wherein the energy of the reservoir is determined by a finite volume conservation equation.

You must specify exactly one convection coefficient, and either exactly one reference temperature or exactly one bulk fluid element name.

33.9.1 Add Surface

Scope: Convective Flux Boundary Condition

```
Add Surface SurfaceList...
```

Parameter	Value	Default
SurfaceList	string...	undefined

Summary Adds surfaces, by name, to a boundary condition's extent.

Description This line command is used to add surfaces to the extent of a boundary condition. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.9.2 Average Temperature Variable

Scope: Convective Flux Boundary Condition

`Average Temperature Variable {=|are|is}` *`NName`*

Parameter	Value	Default
NName	`string`	`undefined`

Summary Specify the name of a global variable to store the average temperature of the specified surfaces for this convective flux command block. This is used if the convective coefficient temperature difference function is set else it is ignored.

33.9.3 Convective Coefficient

Scope: Convective Flux Boundary Condition

`Convective Coefficient {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify a constant convective coefficient for this boundary condition.

33.9.4 Convective Coefficient Encore Function

Scope: Convective Flux Boundary Condition

`Convective Coefficient Encore Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of an Encore function that will be used to calculate the convective coefficient for this boundary condition.

33.9.5 Convective Coefficient Fortran Subroutine

Scope: Convective Flux Boundary Condition

`Convective Coefficient Fortran Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a FORTRAN user subroutine that will be used to calculate the convective coefficient for this boundary condition.

33.9.6 Convective Coefficient Node Variable

Scope: Convective Flux Boundary Condition

`Convective Coefficient Node Variable {=|are|is}` *Name*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of the node variable to use for the convective coefficient that is associated with this boundary condition.

Description The indicated node variable must be a legal Aria variable. Typically, this variable is defined by the user in the user definition command block. The node variable is interpolated to the integration points during the integration of the flux term. Typically, this variable would be calculated in another SIERRA region, e.g., Fuego, and transferred to Aria.

33.9.7 Convective Coefficient Scale Factor

Scope: Convective Flux Boundary Condition

`Convective Coefficient Scale Factor {=|are|is}` *Magnitude*

Parameter	Value	Default
Magnitude	`real`	`undefined`

Summary Specify that the convective coefficient is to be computed using sideset distribution factors that must be defined in the mesh database.

Description The coefficient on each face is computed by interpolating the distribution factors to the Gauss points and then multiplying the result by the given magnitude. Distribution factors do not currently work with h–adaptivity, since their values are not currently interpolated to the new nodes.

33.9.8 Convective Coefficient Subroutine

Scope: Convective Flux Boundary Condition

`Convective Coefficient Subroutine {=|are|is}` *Name*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a user subroutine that is to be used to calculate the convective coefficient for this boundary condition.

If the user subroutine employs a user specified reference temperature model to compute the heat transfer coefficient then optional arguments USING_REF_TEMP must also appear at the end of the command line.

33.9.9 Convective Coefficient Temperature Difference Function

Scope: Convective Flux Boundary Condition

`Convective Coefficient Temperature Difference Function {=|are|is}` *`NName`*

Parameter	Value	Default
NName	`string`	`undefined`

Summary Specify the name of a temperature difference-dependent function that is to be used to calculate the convective coefficient for this boundary condition. If this line command is specified, then a line command for the average temperature must also be specified

33.9.10 Convective Coefficient Temperature Function

Scope: Convective Flux Boundary Condition

`Convective Coefficient Temperature Function {=|are|is}` *`NName`*

Parameter	Value	Default
NName	`string`	`undefined`

Summary Specify the name of a temperature-dependent function that is to be used to calculate the convective coefficient for this boundary condition.

33.9.11 Convective Coefficient Time Function

Scope: Convective Flux Boundary Condition

`Convective Coefficient Time Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a time-dependent function that is used to calculate the convective coefficient for this boundary condition.

33.9.12 Equation

Scope: Convective Flux Boundary Condition

`Equation {=|are|is}` *`EquationName`*

Parameter	Value	Default
EquationName	`string`	`undefined`

Summary This command can be used to apply the Convective Flux conditions for an equation other than the ENERGY equation.

33.9.13 Ignore Flux Coverage

Scope: Convective Flux Boundary Condition

Summary This command causes the code to ignore the flux coverage Field when contact is present. Thus the flux will be applied even if the BC is being set on a contact surface.

33.9.14 Integer Data

Scope: Convective Flux Boundary Condition

`Integer Data` *`Values...`*

Parameter	Value	Default
Values	`integer...`	`undefined`

Summary List of integer data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.9.15 Integrated Flux Output

Scope: Convective Flux Boundary Condition

`Integrated Flux Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Calculate the average flux associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power. This power is then divided by the total area of the surface to obtain the average flux on the surface, and stored in a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.9.16 Integrated Power Output

Scope: Convective Flux Boundary Condition

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Calculate the total power associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.9.17 Real Data

Scope: Convective Flux Boundary Condition

`Real Data` *`Values...`*

Parameter	Value	Default
Values	`real...`	`undefined`

Summary List of real data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.9.18 Ref Temp Convective Coefficient Subroutine

Scope: Convective Flux Boundary Condition

`Ref Temp Convective Coefficient Subroutine {=|are|is}` *`Name`* `[` *`UsingRefTemp`* `]`

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a user subroutine that is to be used to calculate the convective coefficient for this boundary condition. Use of this command line will enable the reference temperature to be supplied to the subroutine interface.

33.9.19 Reference Temperature

Scope: Convective Flux Boundary Condition

`Reference Temperature {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify a constant reference temperature for this boundary condition.

33.9.20 Reference Temperature Fortran Subroutine

Scope: Convective Flux Boundary Condition

`Reference Temperature Fortran Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a FORTRAN user-defined subroutine that will be used to calculate the reference temperature associated with this boundary condition.

33.9.21 Reference Temperature Global Variable

Scope: Convective Flux Boundary Condition

`Reference Temperature Global Variable {=|are|is}` *`GlobalVariableName`*

Parameter	Value	Default
GlobalVariableName	`string`	`undefined`

Summary Specify a global variable to be used for reference temperature.

33.9.22 Reference Temperature Node Variable

Scope: Convective Flux Boundary Condition

`Reference Temperature Node Variable {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of the node variable to use for the reference temperature that is associated with this boundary condition.

Description The indicated node variable must be a legal Aria variable. Typically, this variable is defined by the user in the user definition command block. The node variable is interpolated to the integration points during the integration of the flux term. Typically, this variable would be calculated in another SIERRA region, e.g., Fuego, and transferred to Aria.

33.9.23 Reference Temperature Subroutine

Scope: Convective Flux Boundary Condition

`Reference Temperature Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a user-defined subroutine that is to be used to calculate the reference temperature associated with this boundary condition.

33.9.24 Reference Temperature Temperature Function

Scope: Convective Flux Boundary Condition

`Reference Temperature Temperature Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of the temperature-dependent function that is to be used to calculate the reference temperature associated with this boundary condition.

33.9.25 Reference Temperature Time Function

Scope: Convective Flux Boundary Condition

`Reference Temperature Time Function {=|are|is}` *`FunctionName`*

Parameter	Value	Default
FunctionName	string	undefined

Summary Specify the name of a time-dependent function for the reference temperature for this boundary condition.

33.9.26 Scaled Convective Coefficient Subroutine

Scope: Convective Flux Boundary Condition

`Scaled Convective Coefficient Subroutine {=|are|is}` *`Name FieldName`* `{=|are|is}` *`Field`*

Parameter	Value	Default
Name	string	undefined
FieldName	string	undefined
Field	string	undefined

Summary Specify the name of a user subroutine that is to be used to calculate the scaled convective coefficient for this boundary condition. The interpolated scaling Field is supplied directly to the user subroutine.

33.9.27 Scaled Ref Temp Convective Coefficient Subroutine

Scope: Convective Flux Boundary Condition

`Scaled Ref Temp Convective Coefficient Subroutine {=|are|is}` *`Name FieldName`* `{=|are|is}` *`Field`*

Parameter	Value	Default
Name	string	undefined
FieldName	string	undefined
Field	string	undefined

Summary Specify the name of a user subroutine that is to be used to calculate the scaled convective coefficient for this boundary condition. Use of this command will enable the reference temperature to be supplied to the user subroutine. Additionally interpolated values of the scaling Field will be delivered to the subroutine.

33.9.28 Uq Flux Multiplier

Scope: Convective Flux Boundary Condition

`Uq Flux Multiplier {=|are|is}` *`Value`*

Parameter	Value	Default
Value	real	1.0

Summary Specify constant scaling of the convective flux.

Description Intended use of this scaling parameter is primarily for evaluation of model sensitivities.

33.9.29 Use Advective Bar

Scope: Convective Flux Boundary Condition

`Use Advective Bar` *`Name`* `[` *`BulkNodes`* `]`

Parameter	Value	Default
Name	`string`	`undefined`

Summary Use the named advective bar to model the reference temperature.

Description This line command specifies the name of an advective bar that has been defined using the advective bar command block. The temperature of the advective bar is used as the reference temperature, T_r, for the convective heat transfer and this is computed based on a geometric coupling algorithm. Note that it is illegal to specify both a reference temperature and an advective bar.

33.9.30 Use Bulk Element

Scope: Convective Flux Boundary Condition

`Use Bulk Element` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Use the named bulk element to model the reference temperature.

Description This line command specifies the name of a bulk fluid element that has been defined using the bulk fluid command block. The temperature of the bulk fluid element is used as the reference temperature, T_r, for the convective heat transfer. Note that it is illegal to specify both a reference temperature and a bulk element.

33.9.31 Use Correlation Convection Model

Scope: Convective Flux Boundary Condition

`Use Correlation Convection Model` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies correlation model for convection coefficient

33.9.32 Use Data Block

Scope: Convective Flux Boundary Condition

`Use Data Block` *Name*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

33.9.33 Use Enclosure

Scope: Convective Flux Boundary Condition

`Use Enclosure` *Name*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Use the named enclosure to define the surface list and possibly model the reference temperature using the MBL bulk node of the enclosure.

Description — This line command specifies the name of an enclosure that has been defined using the enclosure definition command block. The list of surfaces for this convective BC is taken from the enclosure surfaces and it is illegal to specify both an enclosure and a surface list. If the enclosure has the Mean Beam Length (MBL) model activated, then this convective BC is linked to the bulk fluid element associated with the MBL model. The temperature of the bulk fluid element is used as the reference temperature, T_r, for the convective heat transfer. Note that it is illegal to specify both a reference temperature and an enclosure with an MBL bulk element.

33.9.34 Use Toggle Block

Scope: Convective Flux Boundary Condition

`Use Toggle Block` *ToggleName* `[ {@|at|for|in|on|over}` *ElementBlockList...* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary — Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.9.35 User Field Mask

Scope: Convective Flux Boundary Condition

`User Field Mask {=|are|is}` *Name* `[` *Threshold* `{=|are|is}` *Value* `]`

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Specify that the convective boundary condition is to be masked by nonzero values of a user defined Field.

Description — The convective boundary condition is applied in a conventional manner but then masked by nonzero values of an interpolated user defined Field. In most cases the user Field is transferred to Aria but it could also be computed. Default threshold for the masked value is 0.0, i.e. mask value equals 1.0 for values greater than zero.

33.9.36 User Field Scaling

Scope: Convective Flux Boundary Condition

`User Field Scaling {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Specify that the convective boundary condition is to be scaled by a user defined Field.

Description — The convective boundary condition is applied in a conventional manner but then scaled by an interpolated user defined Field. In most cases the user Field is transferred to Aria and acts as a weighting of the flux condition.

33.10 Aerodynamic Convection Heat Transfer

High-speed flows about bodies are often characterized by large temperature gradients and large variations of the flow properties through the boundary layer. The problem of heat transfer about the body is often addressed using numerical techniques of CFD to resolve the thermal transport through the boundary layer. Because CFD resources may not be readily applied at the design stage researchers often apply simplifying assumptions to arrive at alternative models for solving the problem of heat transfer about bodies exposed to high-speed flow. One common modeling approach is to define a convection coefficient and reference temperature that depend upon flight conditions. The aero heat flux boundary condition provides a means for supplying details of the flight conditions to arrive at a representative convection coefficient and reference temperature for two heat flux models, one derived by [41] and one that varies only with altitude.

The model by Eckert suggests that a convective heat transfer approach ignoring boundary layer temperature gradients may be used if the properties are evaluated at an alternative reference temperature representative of the averaged flow conditions. Syntax for typical usage of this model is shown below.

```
Begin AERO HEAT FLUX BOUNDARY CONDITION body_heating
  START TIME =  0.0
  STOP TIME =  20.0
  freestream temperature = 233.15
  freestream pressure    = 5.066e3
  mach number = 3.0
  fluid viscosity temperature function           = air_viscosity
  fluid specific heat temperature function       = air_specheat
```

```
  fluid thermal conductivity temperature function = air_cond
  fluid gamma = 1.4
  adiabatic wall temperature from recovery factor
  eckert convective coefficient
  COORDINATE OFFSET AXIS = x
  cone length = 0.0
  add surface surface_1
  integrated power output total_heating
End
```

Another aerodynamic heat flux model defines a convective coefficient that relies upon the variation of density with altitude. Syntax for typical usage of this model is shown below.

```
Begin AERO HEAT FLUX BOUNDARY CONDITION body_heating
  START TIME =  0.0
  STOP TIME =  20.0
  Density Ratio Convective Coefficient
  Freestream Density Altitude Function = density_func
  Altitude time function = altitude_time_func
  Reference Density = 1.15
  Reference HTC = 25.0
  Density Ratio exponent = 0.6
  reference temperature time function = ref_temp_func
  add surface surface_1
  integrated power output total_heating
End
```

A more complete description of syntax for the aero heating models is described in the section which follows.

33.11 Aero Heat Flux Boundary Condition

Scope: Equation System

Begin Aero Heat Flux Boundary Condition *Name*

Add Surface *SurfaceList...*

Adiabatic Wall Temperature {= | are | is} *Value*

Adiabatic Wall Temperature Altitude Function {= | are | is} *Function*

Adiabatic Wall Temperature From Recovery Factor

Adiabatic Wall Temperature Time Function {= | are | is} *Function*

Altitude {= | are | is} *Value*

Altitude Time Function {= | are | is} *Function*

Cone Length {= | are | is} *Value*

Coordinate Offset {= | are | is} *Value*

Coordinate Offset Axis {= | are | is} *axis*

Density Ratio Convective Coefficient

Density Ratio Exponent {= | are | is} *Value*

```
    Eckert Convective Coefficient
    Fluid Gamma {=|are|is} Value
    Fluid Gas Constant {=|are|is} Value
    Fluid Properties Temperature Function {=|are|is} Function
    Fluid Specific Heat Temperature Function {=|are|is} Function
    Fluid Thermal Conductivity Temperature Function {=|are|is} Function
    Fluid Viscosity Temperature Function {=|are|is} Function
    Freestream Density Altitude Function {=|are|is} Function
    Freestream Pressure {=|are|is} Value
    Freestream Pressure Altitude Function {=|are|is} Function
    Freestream Pressure Time Function {=|are|is} Function
    Freestream Temperature {=|are|is} Value
    Freestream Temperature Altitude Function {=|are|is} Function
    Freestream Temperature Time Function {=|are|is} Function
    Integrated Flux Output VariableName
    Integrated Power Output VariableName
    Mach Number {=|are|is} Value
    Mach Number Time Function {=|are|is} Function
    Reference Density {=|are|is} Value
    Reference Htc {=|are|is} Value
    Reference Temperature {=|are|is} Value
    Reference Temperature Time Function {=|are|is} FunctionName
    Start Time {=|are|is} Value
    Stop Time {=|are|is} Value
    Uq Flux Multiplier {=|are|is} Value
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList...  ]
End
```

Summary — This command block defines a distributed convective heat flux about a body based upon aerodynamic flight conditions applied on the given surface.

Description — The aerodynamic heat flux depends upon characterization of flight conditions prescribed using data tables and spatial location on the body of interest.

33.11.1 Add Surface

Scope: Aero Heat Flux Boundary Condition

```
Add Surface SurfaceList...
```

Parameter	Value	Default
SurfaceList	string...	undefined

Summary Adds surfaces, by name, to a boundary condition's extent.

Description This line command is used to add surfaces to the extent of a boundary condition. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.11.2 Adiabatic Wall Temperature

Scope: Aero Heat Flux Boundary Condition

`Adiabatic Wall Temperature {= | are | is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary Specifies a constant adiabatic wall temperature. Usage: ECKERT CONVECTIVE COEFFICIENT.

FREESTREAM TEMPERATURE and fluid properties consistent with this ADIABATIC WALL TEMPERATURE must also be provided.

33.11.3 Adiabatic Wall Temperature Altitude Function

Scope: Aero Heat Flux Boundary Condition

`Adiabatic Wall Temperature Altitude Function {= | are | is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the adiabatic wall temperature versus altitude function for climate condition. Usage: ECKERT CONVECTIVE COEFFICIENT.

FREESTREAM TEMPERATURE and fluid properties consistent with this ADIABATIC WALL TEMPERATURE must also be provided.

33.11.4 Adiabatic Wall Temperature From Recovery Factor

Scope: Aero Heat Flux Boundary Condition

Summary Specifies that the adiabatic wall temperature be computed using the recovery factor, the stagnation temperature and the freestream temperature. Usage: ECKERT CONVECTIVE COEFFICIENT.

Description The adiabatic wall temperature T_{aw} is computed as

$$T_{aw} = rT_o + (1 - r)T_\infty$$

where r is the recovery factor, T_o is the stagnation temperature and T_∞ is the freestream temperature. The user must provide specification of MACH NUMBER, FREESTREAM

TEMPERATURE, FREESTREAM PRESSURE and FLUID_GAMMA models. Stagnation temperature and flow velocity v will be internally computed from the MACH NUMBER. The recovery factor is given in terms of the freestream Prandtl number ($C_p\mu/K$) as

$$r = \begin{cases} Pr^{1/2} & \text{laminar flow} \\ Pr^{1/3} & \text{turbulent flow } Re > 5X10^5 \end{cases} .$$

Hence one must provide functions for specific heat, dynamic viscosity and thermal conductivity as a function of temperature. The flow regime varies spatially in the COORDINATE OFFSET direction as determined using the local Reynolds number

$$Re_x = \frac{\rho v x}{\mu} .$$

Here the pressure P will vary with altitude hence density is evaluated using the ideal gas law

$$\rho = \frac{P}{RT_\infty}$$

33.11.5 Adiabatic Wall Temperature Time Function

Scope: Aero Heat Flux Boundary Condition

`Adiabatic Wall Temperature Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary

Specifies the name of adiabatic wall temperature versus time function. Usage: ECKERT CONVECTIVE COEFFICIENT.

FREESTREAM TEMPERATURE and fluid properties consistent with this ADIABATIC WALL TEMPERATURE must also be provided.

33.11.6 Altitude

Scope: Aero Heat Flux Boundary Condition

`Altitude {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary

Specifies a constant altitude.

Used in conjunction with FREESTREAM TEMPERATURE ALTITUDE FUNCTION, FREESTREAM TEMPERATURE FUNCTION and ECKERT CONVECTIVE COEFFICIENT. Alternative usage with FREESTREAM DENSITY ALTITUDE FUNCTION and DENSITY RATIO CONVECTIVE COEFFICIENT.

33.11.7 Altitude Time Function

Scope: Aero Heat Flux Boundary Condition

`Altitude Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the altitude versus time function for a given flight. The tabulated time is defined relative to the START TIME.

Used in conjunction with FREESTREAM TEMPERATURE ALTITUDE FUNCTION, FREESTREAM TEMPERATURE FUNCTION and ECKERT CONVECTIVE COEFFICIENT. Alternative usage with FREESTREAM DENSITY ALTITUDE FUNCTION and DENSITY RATIO CONVECTIVE COEFFICIENT.

33.11.8 Cone Length

Scope: Aero Heat Flux Boundary Condition

`Cone Length {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary Specifies the length at which the body profile transitions to a constant diameter. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.9 Coordinate Offset

Scope: Aero Heat Flux Boundary Condition

`Coordinate Offset {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary Specifies the coordinate offset from which the spatial position will be defined in the ECKERT CONVECTIVE COEFFICIENT calculations $x = X - \text{offset}$ where X is the Cartesian model coordinate from the mesh prescribed using COORDINATE OFFSET AXIS and is assumed to define the leading edge. The value of x is used in computation of the Reynolds number $Re_x^* = \frac{\rho^* v x}{\mu}$ hence the evaluation of x must provide values greater than zero.

33.11.10 Coordinate Offset Axis

Scope: Aero Heat Flux Boundary Condition

`Coordinate Offset Axis {=|are|is}` *`axis`*

Parameter	Value	Default
axis	`string`	`none`

Summary Specifies the Cartesian coordinate axis (X, Y or Z) from which spatial position will be defined when using the ECKERT CONVECTIVE COEFFICIENT. The surface heat transfer coefficient will be computed spatially relative to zero on this axis.

For models in which the geometry is offset from a zero reference one must define the COORDINATE OFFSET.

33.11.11 Density Ratio Convective Coefficient

Scope: Aero Heat Flux Boundary Condition

Summary — Specifies a density ratio heat transfer coefficient model. Requires that FREESTREAM DENSITY ALTITUDE FUNCTION, REFERENCE DENSITY, REFERENCE HTC, DENSITY RATIO EXPONENT, START TIME and STOP TIME specifications be supplied.

Description — The heat transfer coefficient that varies as

$$h = h_o \left[\frac{\rho(A)}{\rho_o} \right]^n$$

where h_o is the reference heat transfer coefficient, $\rho(A)$ is the altitude dependent density, ρ_o is the reference density and n is the density ratio exponent. Here the model parameters h_o, ρ_o and n are calibrated based upon measured temperature data.

33.11.12 Density Ratio Exponent

Scope: Aero Heat Flux Boundary Condition

`Density Ratio Exponent {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary — Specifies the density ratio exponent value for the density ratio heat transfer coefficient model. Usage: DENSITY RATIO CONVECTIVE COEFFICIENT.

33.11.13 Eckert Convective Coefficient

Scope: Aero Heat Flux Boundary Condition

Summary — Specifies use of the Eckert heat transfer coefficient model for high-speed flow. Requires that freestream FLUID PROPERTY TEMPERATURE FUNCTIONs, START TIME, STOP TIME, FREESTREAM PRESSURE and ADIABATIC WALL TEMPERATURE specifications be supplied.

Description — A constant property heat transfer coefficient approach in which properties are evaluated at an alternative temperature

$$T^* = T_\infty + 0.5(T - T_\infty) + 0.22(T_{aw} - T_\infty)$$

where T_∞ is the freestream temperature, T is the surface temperature, T_{aw} is the adiabatic wall temperature. T_{aw} will vary spatially depending upon the local Reynolds number

$$Re_x^* = \frac{\rho^* v x}{\mu}$$

where μ is evaluated at T^* and the coordinate $x > 0$. The FREESTREAM PRESSURE varies with altitude hence the density ρ^* is evaluated using the ideal gas law

$$\rho^* = \frac{P}{RT^*} \ .$$

The Stanton number represents the ratio of heat transferred into the fluid flow to the thermal capacity of the fluid. By developing the relationship between fluid friction and heat transfer frictional resistance, the heat transfer can be expressed in terms of the Stanton number to arrive at an approximate heat transfer coefficient for various portions of the flow regime

$$St_x^* Pr^{*2/3} = \begin{cases} 0.332\ (Re_x^*/f_M)^{-1/2} & Re_x^* < 5 \times 10^5 & \text{laminar} \\ 0.0296(Re_x^*/f_M)^{-1/5} & 5 \times 10^5 < Re_x^* < 10^7 & \text{turbulent} \\ 0.185\ [log10(Re_x^*/f_M)]^{-2.584} & 10^7 < Re_x^* < 10^9 & \text{turbulent} \\ 0.00063 & Re_x^* > 10^9 & \text{turbulent} \end{cases}$$

The parameter f_M is the Mangler transformation for cone geometry

$$f_M = \begin{cases} 3 & \text{cone geometry} & \text{laminar} \\ 2 & \text{cone geometry} & \text{turbulent} \\ 1 & \text{flat plate geometry} & \text{laminar} \end{cases}$$

While Eckert's approach considers an average heat transfer coefficient over portions of a surface, here we consider a local heat transfer coefficient, $h(x)$ based upon the above relations that can be applied directly to the surface discretization

$$h(x) = \rho^* C_p^* U_\infty St_x^*$$

33.11.14 Fluid Gamma

Scope: Aero Heat Flux Boundary Condition

`Fluid Gamma {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`1.4`

Summary Specifies a constant specific heat ratio of the fluid medium. Usage: ECKERT CONVECTIVE COEFFICIENT.

Only constant values of γ are allowed.

33.11.15 Fluid Gas Constant

Scope: Aero Heat Flux Boundary Condition

`Fluid Gas Constant {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`none`

Summary Specifies the gas constant of the fluid medium. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.16 Fluid Properties Temperature Function

Scope: Aero Heat Flux Boundary Condition

`Fluid Properties Temperature Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	string	undefined

Summary Specifies a multi-column function containing fluid properties density, specific heat, conductivity and dynamic viscosity versus temperature with respective property columns named CP, K and MU. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.17 Fluid Specific Heat Temperature Function

Scope: Aero Heat Flux Boundary Condition

`Fluid Specific Heat Temperature Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	string	undefined

Summary Specifies the name of a user tabular function for specific heat as a function of temperature. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.18 Fluid Thermal Conductivity Temperature Function

Scope: Aero Heat Flux Boundary Condition

`Fluid Thermal Conductivity Temperature Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	string	undefined

Summary Specifies the name of a user tabular function for thermal conductivity as a function of temperature. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.19 Fluid Viscosity Temperature Function

Scope: Aero Heat Flux Boundary Condition

`Fluid Viscosity Temperature Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	string	undefined

Summary Specifies the name of a user tabular function for dynamic viscosity as a function of temperature. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.20 Freestream Density Altitude Function

Scope: Aero Heat Flux Boundary Condition

`Freestream Density Altitude Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	string	undefined

Summary Specifies the freestream density versus altitude function for climate condition. Usage: DENSITY RATIO CONVECTIVE COEFFICIENT.

Requires that one supply an ALTITUDE model.

33.11.21 Freestream Pressure

Scope: Aero Heat Flux Boundary Condition

`Freestream Pressure {= | are | is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary Specifies a constant freestream pressure.

33.11.22 Freestream Pressure Altitude Function

Scope: Aero Heat Flux Boundary Condition

`Freestream Pressure Altitude Function {= | are | is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the freestream pressure versus altitude function for climate condition.

33.11.23 Freestream Pressure Time Function

Scope: Aero Heat Flux Boundary Condition

`Freestream Pressure Time Function {= | are | is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the name of freestream pressure versus time function.

33.11.24 Freestream Temperature

Scope: Aero Heat Flux Boundary Condition

`Freestream Temperature {= | are | is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary Specifies a constant freestream temperature. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.25 Freestream Temperature Altitude Function

Scope: Aero Heat Flux Boundary Condition

`Freestream Temperature Altitude Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the freestream temperature versus altitude function for climate condition. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.26 Freestream Temperature Time Function

Scope: Aero Heat Flux Boundary Condition

`Freestream Temperature Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the name of freestream temperature versus time function. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.27 Integrated Flux Output

Scope: Aero Heat Flux Boundary Condition

`Integrated Flux Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Calculate the average flux associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power. This power is then divided by the total area of the surface to obtain the average flux on the surface, and stored in a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.11.28 Integrated Power Output

Scope: Aero Heat Flux Boundary Condition

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Calculate the total power associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.11.29 Mach Number

Scope: Aero Heat Flux Boundary Condition

`Mach Number {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary Specifies a constant Mach number. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.30 Mach Number Time Function

Scope: Aero Heat Flux Boundary Condition

`Mach Number Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the Mach number versus time function for a given flight. The tabulated time is defined relative to the START TIME. Usage: ECKERT CONVECTIVE COEFFICIENT.

33.11.31 Reference Density

Scope: Aero Heat Flux Boundary Condition

`Reference Density {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies the reference density value for the density ratio heat transfer coefficient model. Usage: DENSITY RATIO CONVECTIVE COEFFICIENT.

33.11.32 Reference Htc

Scope: Aero Heat Flux Boundary Condition

`Reference Htc {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies the reference heat transfer coefficient value for the density ratio heat transfer coefficient model. Usage: DENSITY RATIO CONVECTIVE COEFFICIENT.

33.11.33 Reference Temperature

Scope: Aero Heat Flux Boundary Condition

`Reference Temperature {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify a constant reference temperature for this boundary condition.

33.11.34 Reference Temperature Time Function

Scope: Aero Heat Flux Boundary Condition

`Reference Temperature Time Function {=|are|is}` *`FunctionName`*

Parameter	Value	Default
FunctionName	`string`	`undefined`

Summary Specify the name of a time-dependent function for the reference temperature for this boundary condition.

33.11.35 Start Time

Scope: Aero Heat Flux Boundary Condition

`Start Time {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`0.0`

Summary Specifies the analysis time at which the flight begins and the AERO heat flux will be applied.

33.11.36 Stop Time

Scope: Aero Heat Flux Boundary Condition

`Stop Time {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`none`

Summary Specifies the analysis time at which the flight ends and the AERO heat flux BC becomes inactive.

33.11.37 Uq Flux Multiplier

Scope: Aero Heat Flux Boundary Condition

`Uq Flux Multiplier {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`1.0`

Summary Specify constant scaling of the convective flux.

Description Intended use of this scaling parameter is primarily for evaluation of model sensitivities.

33.11.38 Use Toggle Block

Scope: Aero Heat Flux Boundary Condition

```
Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList... ]
```

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.12 Advective Bar

Beta Capability: The advective bar feature has not been fully tested and optimized for production use. Use this capability with caution.

For problems in which convection heat transfer occurs at the surface the analyst must supply some judgment about the fluid flow that gives rise to convection. While this information usually by performing a coupled conjugate heat transfer analysis including details of the fluid flow. In many cases the cost of such an analysis may be prohibitive so the analyst must resort to alternative lower order models. One such model, the advective bar, can be applied to cases in which the convective heat transfer is driven by a flow internal to the body of interest as in Figure 33.1. The bar model includes a simplified description of forced flow and flow section along the length of a one-dimensional discretization, where the advected flow is directly coupled with a corresponding energy equation. The bar temperatures define the far-field temperature distribution for convective heat transfer to the two or three-dimensional thermal solid model. The advantage gained by using the simplified velocity and temperature representation is that the velocity can be used to define both laminar and turbulent convective heat transfer appropriate to the specific problem using empirical heat transfer coefficients 34.1.

A bar model coordinate system facilitates user specification of the flow section properties. Here the bar coordinate system can be aligned with one of a Cartesian coordinate X,Y,Z directions. Optional use of an S coordinate system tied directly to the bar discretization enables modelling of flow along a circuitous path.

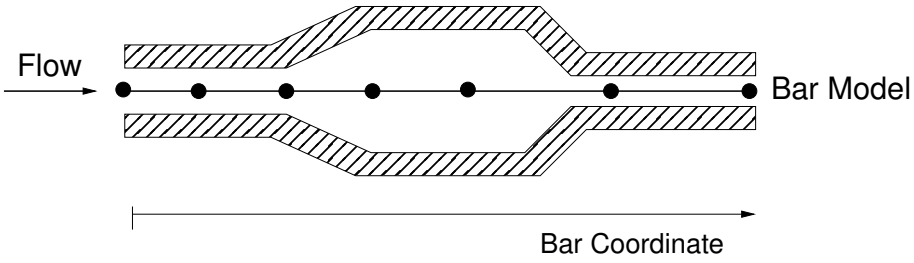


Figure 33.1. Interior Body Flow

Entrance length effects are supported for a limited number of empirical correlations. By default, influence of the entrance length effect is aligned with the bar model coordinates. Cases in which these effects are significant only over a portion of the flow can be modeled using an offset of the bar model coordinate 33.2.

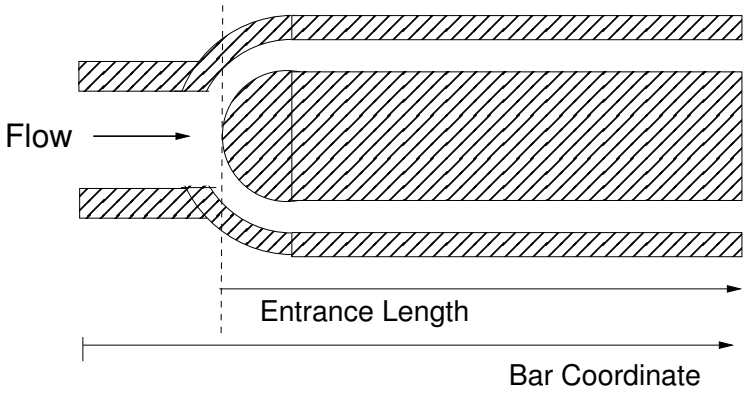


Figure 33.2. Flow Entrance Effect Coordinate

Use of the advective bar model requires that it be referenced by name in a Convective Flux Boundary Condition along with the corresponding named Heat Transfer Coefficient Correlation.

```
Begin Sierra myJob
  .
  Begin Procedure My_Aria_Procedure
    .
    Begin Aria Region My_Region
      .
      Begin ADVECTIVE bAR NETWORK airBar
        .
      End ADVECTIVE BAR NETWORK airBar
      .
      Begin HEAT TRANSFER CORRELATION COEFFICIENT channel
        .
      End HEAT TRANSFER CORRELATION COEFFICIENT channel
      .
      begin convective flux boundary condition inside
        .
        USE CORRELATION CONVECTION MODEL channel
```

```
        USE ADVECTIVE BAR airbar
        .
      end convective flux boundary condition inside
     .
    End
    .
  End
  .
End Sierra myJob
```

33.13 Advective Bar Network

Scope: Equation System

```
Begin Advective Bar Network Name
    Add Volume Parameters...
    Annulus Diameter Ratio Function {=|are|is} Function
    Annulus Diameter Ratio Threshold {=|are|is} Threshold
    Coordinate Reference {=|are|is} Nodelist Component xyzs
    Entrance Effect Starting Node Id {=|are|is} NodeID [ Offset {=|are|is} Value ]
    Flow Cross Sectional Area Function {=|are|is} Function
    Fluid Density {=|are|is} Value
    Fluid Density Function {=|are|is} Function
    Fluid Viscosity {=|are|is} Value
    Hydraulic Diameter Function {=|are|is} Function
    Initial Temperature {=|are|is} t
    Mass Flow Rate {@|at|for|in|on|over} Nodelist {=|are|is} Value
    Mass Flow Rate Function {@|at|for|in|on|over} Nodelist {=|are|is} Function
    Maximum Pressure Solve Iterations {=|are|is} Iterations
    Maximum Search Tolerance {=|are|is} Tolerance
    Pressure Bc {@|at|for|in|on|over} Nodelist {=|are|is} Pres
    Pressure Solve Tolerance {=|are|is} Tolerance
    Search Interval {=|are|is} Interval
    Solve Pressure
    Surface Roughness {=|are|is} Roughness
    Velocity Bc {@|at|for|in|on|over} Nodelist {=|are|is} Velocity
    Visualization OnOff
    Wetted Perimeter Function {=|are|is} Function
End
```

Summary Defines an advective bar that can be used by a convective flux boundary condition.

Description Newton's law of cooling specifies that the heat flux normal to a surface is proportional to the difference between the unknown temperature of the surface and some reference temperature of the fluid in which the surface is immersed: $q_n = h(T - T_r)$. The convection coefficient, h, and the reference temperature, T_r, may be specified in several ways. Of particular interest is the situation where the fluid flow can be reduced to a simple 1D flow. In this case, heat transfer between the surface and the fluid is modeled with an advective bar model where each segment on the bar and each facet on the surface assumes a constant heat flux based on a constant h, T and T_r.

33.13.1 Add Volume

Scope: Advective Bar Network

`Add Volume` *`Parameters...`*

Parameter	Value	Default
Parameters	`string...`	`undefined`

Summary Adds bar element volumes, by name, to a boundary condition's extent.

33.13.2 Annulus Diameter Ratio Function

Scope: Advective Bar Network

`Annulus Diameter Ratio Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the name of a user tabular function for annulus diameter ratio.

33.13.3 Annulus Diameter Ratio Threshold

Scope: Advective Bar Network

`Annulus Diameter Ratio Threshold {=|are|is}` *`Threshold`*

Parameter	Value	Default
Threshold	`real`	`0.05`

Summary Specifies the minimum value for annulus diameter ratio. This is used with both automatic calculation as well as user tabular function.

33.13.4 Coordinate Reference

Scope: Advective Bar Network

`Coordinate Reference {=|are|is}` *`Nodelist`* `Component` *`xyzs`*

Parameter	Value	Default
Nodelist	`string`	`undefined`
xyzs	`{s|x|y|z}`	`undefined`

Summary Specifies the reference nodelist and direction for bar coordinates.

33.13.5 Entrance Effect Starting Node Id

Scope: Advective Bar Network

`Entrance Effect Starting Node Id {=|are|is}` *`NodeID`* `[` *`Offset`* `{=|are|is}` *`Value`* `]`

Parameter	Value	Default
NodeID	`integer`	0

Summary Specifies the node in the advective bar that corresponds to the position where entrance effects begin with pipe and annular flow correlations (Types 23, 25, 72-74).

Description Pipe and annular flow correlations support an entrance effect correction factor that will adjust the local heat transfer coefficient based on how far a point is from the inlet of the pipe or annulus. By default the inlet of the pipe is assumed to be at the first node of the advective bar. This option allows the user to specify a different node ID as the pipe entrance for calculating the entrance effect. This node ID must be part of the advective bar network.

Optionally, in cases where the entrance effect begins at a point not exactly aligned with the STARTING NODE one may specify an offset that will align the entrance effect to the correct coordinate. Here the offset is defined in the COORDINATE REFERENCE direction.

33.13.6 Flow Cross Sectional Area Function

Scope: Advective Bar Network

`Flow Cross Sectional Area Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the name of a user tabular function for cross-sectional flow area.

33.13.7 Fluid Density

Scope: Advective Bar Network

`Fluid Density {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies bar fluid density.

33.13.8 Fluid Density Function

Scope: Advective Bar Network

`Fluid Density Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the name of a user tabular function of temperature for density.

33.13.9 Fluid Viscosity

Scope: Advective Bar Network

`Fluid Viscosity {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies bar fluid absolute viscosity.

33.13.10 Hydraulic Diameter Function

Scope: Advective Bar Network

`Hydraulic Diameter Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the name of a user tabular function for hydraulic diameter.

33.13.11 Initial Temperature

Scope: Advective Bar Network

`Initial Temperature {=|are|is}` *`t`*

Parameter	Value	Default
t	`real`	`undefined`

Summary Specifies the initial temperature of the bulk node.

33.13.12 Mass Flow Rate

Scope: Advective Bar Network

`Mass Flow Rate {@|at|for|in|on|over}` *`Nodelist`* `{=|are|is}` *`Value`*

Parameter	Value	Default
Nodelist	`string`	`undefined`
Value	`real`	`undefined`

Summary Specifies the mass flow rate at a node.

33.13.13 Mass Flow Rate Function

Scope: Advective Bar Network

`Mass Flow Rate Function {@|at|for|in|on|over}` *`Nodelist`* `{=|are|is}` *`Function`*

Parameter	Value	Default
Nodelist	`string`	`undefined`
Function	`string`	`undefined`

Summary Specifies the mass flow rate as a function of time at a node with a user function.

33.13.14 Maximum Pressure Solve Iterations

Scope: Advective Bar Network

`Maximum Pressure Solve Iterations {=|are|is}` *`Iterations`*

Parameter	Value	Default
Iterations	`integer`	`undefined`

Summary Specifies maximum number of bar pressure solve iterations.

33.13.15 Maximum Search Tolerance

Scope: Advective Bar Network

`Maximum Search Tolerance {=|are|is}` *`Tolerance`*

Parameter	Value	Default
Tolerance	`real`	`0.5`

Summary Specifies the maximum search tolerance for bar/facet geometric coupling search and this must be between 0 and 1. For an idealized case of a curved bar with a constant curvature (R) and a facet that is at a distance (d) from the bar, the analytical solution tolerance is given by

$$tol = \frac{d}{2R}$$

Note that for a straight bar with infinite radius of curvature, this gives a tolerance of 0.

33.13.16 Pressure Bc

Scope: Advective Bar Network

`Pressure Bc {@|at|for|in|on|over}` *`Nodelist`* `{=|are|is}` *`Pres`*

Parameter	Value	Default
Nodelist	`string`	`undefined`
Pres	`real`	`undefined`

Summary Specifies bar inlet pressure.

33.13.17 Pressure Solve Tolerance

Scope: Advective Bar Network

`Pressure Solve Tolerance {=|are|is}` *`Tolerance`*

Parameter	Value	Default
Tolerance	`real`	`undefined`

Summary Specifies bar pressure solution convergence tolerance.

33.13.18 Search Interval

Scope: Advective Bar Network

`Search Interval {=|are|is}` *`Interval`*

Parameter	Value	Default
Interval	`integer`	5

Summary Specifies the number of search intervals for bar/facet geometric coupling search. This is used in conjunction with the maximum search tolerance to generate search tolerances from 0.0 to the maximum search tolerance specified. Must be greater than 0

33.13.19 Solve Pressure

Scope: Advective Bar Network

Summary Invoke pressure solution. This capability is not currently enabled for split network solutions.

33.13.20 Surface Roughness

Scope: Advective Bar Network

`Surface Roughness {=|are|is}` *`Roughness`*

Parameter	Value	Default
Roughness	`real`	`undefined`

Summary Specifies roughness coefficient of surface.

33.13.21 Velocity Bc

Scope: Advective Bar Network

`Velocity Bc {@|at|for|in|on|over}` *`Nodelist`* `{=|are|is}` *`Velocity`*

Parameter	Value	Default
Nodelist	`string`	`undefined`
Velocity	`real`	`undefined`

Summary Specifies bar inlet velocity.

33.13.22 Visualization

Scope: Advective Bar Network

`Visualization` *OnOff*

Parameter	Value	Default
OnOff	{off | on}	Off

Summary Specifies Exodus output toggling of the created subsetted surfaces which are directly affected by the bulk nodes on the advective bar.

33.13.23 Wetted Perimeter Function

Scope: Advective Bar Network

`Wetted Perimeter Function {= | are | is}` *Function*

Parameter	Value	Default
Function	string	undefined

Summary Specifies the name of a user tabular function for wetted perimeter.

33.14 Radiative Flux Boundary Condition

Scope: Equation System

`Begin Radiative Flux Boundary Condition` *Name*

`Add Surface` *SurfaceList...*

`Emissivity {= | are | is}` *Value* `[ On` *SurfaceName* `]`

`Emissivity For Bands {= | are | is}` *Emissivity...*

`Emissivity Fortran Subroutine {= | are | is}` *Name*

`Emissivity Function {= | are | is}` *FunctionName* `[ On` *SurfaceName* `]`

`Emissivity Subroutine {= | are | is}` *MySub* `[ On` *SurfaceName* `]`

`Emissivity Time Function {= | are | is}` *FunctionName* `[ On` *SurfaceName* `]`

`Equation {= | are | is}` *EquationName*

`Form Factor Fortran Subroutine {= | are | is}` *Name*

`Ignore Flux Coverage`

`Integer Data` *Values...*

`Integrated Flux Output` *VariableName*

`Integrated Power Output` *VariableName*

`Irradiation Node Variable {= | are | is}` *Name*

`Irradiation Subroutine {= | are | is}` *Name*

```
    Irradiation Time Function {=|are|is} Name
    Radiation Form Factor {=|are|is} Value
    Radiation Form Factor Subroutine {=|are|is} Name
    Radiation Form Factor Temperature Function {=|are|is} Name
    Radiation Form Factor Time Function {=|are|is} Name
    Real Data Values...
    Reference Temperature {=|are|is} Value
    Reference Temperature Fortran Subroutine {=|are|is} Name
    Reference Temperature Global Variable {=|are|is} GlobalVariableName
    Reference Temperature Subroutine {=|are|is} Name
    Reference Temperature Temperature Function {=|are|is} Name
    Reference Temperature Time Function {=|are|is} FunctionName
    Use Banded Wavelength Model ModelName
    Use Bulk Element Name
    Use Data Block Name
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList... ]
End
```

Summary — Radiative heat flux boundary condition. You must specify either a reference temperature or provide a nodal field for irradiation.

Description — This boundary condition models the radiative heat transfer from a specified surface to a surrounding surface. The temperature of the surrounding surface is assumed to have enough thermal mass that its temperature is unaffected by changes in computed temperature, and is therefore known. The heat flux normal to the surface is is given by $q_n = \varepsilon F(\sigma T^4 - G)$, where the incident radiative flux, or irradiation, G, may either be specified directly as a nodal field, or may be specified through a reference temperature, $G = \sigma T_r^4$. The Stefan-Boltzmann constant, σ, is defined in a Global Constants command block, ε is the surface emissivity, and F is the form factor.

With the exception of σ, each of these parameters may vary in one of several ways, as defined below. The emissivity, if left unspecified, is determined locally from the material associated with the element that underlies each face on the surface. In the case of faces that are created as a result of element death, the emissivity is always determined by the underlying element.

33.14.1 Add Surface

Scope: Radiative Flux Boundary Condition

```
Add Surface SurfaceList...
```

Parameter	Value	Default
SurfaceList	string...	undefined

Summary — Adds surfaces, by name, to a boundary condition's extent.

Description This line command is used to add surfaces to the extent of a boundary condition. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.14.2 Emissivity

Scope: Radiative Flux Boundary Condition

`Emissivity {=|are|is}` *`Value`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
Value	`real`	`undefined`

Summary Sets a constant value of emissivity for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

33.14.3 Emissivity For Bands

Scope: Radiative Flux Boundary Condition

`Emissivity For Bands {=|are|is}` *`Emissivity...`*

Parameter	Value	Default
Emissivity	`real...`	`undefined`

Summary Specify the constant emissivities for each wavelength band.

33.14.4 Emissivity Fortran Subroutine

Scope: Radiative Flux Boundary Condition

`Emissivity Fortran Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a FORTRAN user subroutine that will be used to calculate the convective coefficient for this boundary condition.

33.14.5 Emissivity Function

Scope: Radiative Flux Boundary Condition

`Emissivity Function {=|are|is}` *`FunctionName`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
FunctionName	`string`	`undefined`

Summary Sets a emissivity function for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

33.14.6 Emissivity Subroutine

Scope: Radiative Flux Boundary Condition

`Emissivity Subroutine {=|are|is}` *`MySub`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
MySub	`string`	`undefined`

Summary Sets a emissivity user subroutine for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

Also, the software supports using locally scoped user data for most user subroutines, but I haven't figured out a syntax for it here yet. So it is not yet supported. If you need to get data into this subroutine, use the region's "REAL DATA" and "INTEGER DATA" line commands.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

33.14.7 Emissivity Time Function

Scope: Radiative Flux Boundary Condition

`Emissivity Time Function {=|are|is}` *`FunctionName`* `[ On` *`SurfaceName`* `]`

Parameter	Value	Default
FunctionName	`string`	`undefined`

Summary Sets a emissivity function of time for a defined surface. If the optional parameters are not included, then this is the default emissivity for the enclosure. Otherwise it is only applied to the indicated surface.

**Note that if a surface is called out more than once, the emissivity definition it is overwritten: last one in wins.

33.14.8 Equation

Scope: Radiative Flux Boundary Condition

`Equation {=|are|is}` *`EquationName`*

Parameter	Value	Default
EquationName	`string`	`undefined`

Summary This command can be used to apply the Radiative Flux conditions for an equation other than the ENERGY equation.

33.14.9 Form Factor Fortran Subroutine

Scope: Radiative Flux Boundary Condition

`Form Factor Fortran Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Specify the name of a FORTRAN user subroutine that will be used to calculate the form factor for this boundary condition.

33.14.10 Ignore Flux Coverage

Scope: Radiative Flux Boundary Condition

Summary — This command causes the code to ignore the flux coverage Field when contact is present. Thus the flux will be applied even if the BC is being set on a contact surface.

33.14.11 Integer Data

Scope: Radiative Flux Boundary Condition

`Integer Data` *`Values...`*

Parameter	Value	Default
Values	`integer...`	`undefined`

Summary — List of integer data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.14.12 Integrated Flux Output

Scope: Radiative Flux Boundary Condition

`Integrated Flux Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary — Calculate the average flux associated with this flux boundary condition.

Description — This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power. This power is then divided by the total area of the surface to obtain the average flux on the surface, and stored in a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.14.13 Integrated Power Output

Scope: Radiative Flux Boundary Condition

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary — Calculate the total power associated with this flux boundary condition.

Description — This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.14.14 Irradiation Node Variable

Scope: Radiative Flux Boundary Condition

`Irradiation Node Variable {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Specifies the name of the node variable in which the incident radiative flux associated with this boundary condition is stored.

Description — The indicated node variable must be a valid user variable. Typically, this variable is defined by the user within the Region scope. The node variable is interpolated to the integration points during the integration of the flux term. Typically, this variable would be calculated in another SIERRA region, e.g., Fuego, and populated by transfer in the appropriate Region.

33.14.15 Irradiation Subroutine

Scope: Radiative Flux Boundary Condition

`Irradiation Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Specify the name of a user subroutine used to evaluate the irradiation for this boundary condition.

33.14.16 Irradiation Time Function

Scope: Radiative Flux Boundary Condition

`Irradiation Time Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a time-dependent function that is used to calculate the irradiation associated with this boundary condition.

33.14.17 Radiation Form Factor

Scope: Radiative Flux Boundary Condition

`Radiation Form Factor {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies a constant value for the radiation form factor for this boundary condition.

Description For two surfaces, the BC surface (1) and far-field surface (2), ε denoting emissivity, A denoting area, and F_{12} is the view factor.

$$F = \begin{cases} 1 & \text{if } A_2 \gg A_1 \\ F_{12}/[(1-\varepsilon_1)F_{12}+\varepsilon_1] & \text{for } \varepsilon_1 = 1 \text{ and } A_2 \text{ not} \gg A_1 \\ F_{12} & \text{if } \varepsilon_1 = \varepsilon_2 = 1 \end{cases}$$

33.14.18 Radiation Form Factor Subroutine

Scope: Radiative Flux Boundary Condition

`Radiation Form Factor Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a user subroutine that is used to calculate the radiation form factor associated with this boundary condition.

33.14.19 Radiation Form Factor Temperature Function

Scope: Radiative Flux Boundary Condition

`Radiation Form Factor Temperature Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a temperature-dependent function that is used to calculate the radiation form factor associated with this boundary condition.

33.14.20 Radiation Form Factor Time Function

Scope: Radiative Flux Boundary Condition

`Radiation Form Factor Time Function {=|are|is}` *Name*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a time-dependent function that is used to calculate the form factor associated with this boundary condition.

33.14.21 Real Data

Scope: Radiative Flux Boundary Condition

`Real Data` *Values...*

Parameter	Value	Default
Values	`real...`	`undefined`

Summary List of real data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.14.22 Reference Temperature

Scope: Radiative Flux Boundary Condition

`Reference Temperature {=|are|is}` *Value*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specify a constant reference temperature for this boundary condition.

33.14.23 Reference Temperature Fortran Subroutine

Scope: Radiative Flux Boundary Condition

`Reference Temperature Fortran Subroutine {=|are|is}` *Name*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a FORTRAN user-defined subroutine that will be used to calculate the reference temperature associated with this boundary condition.

33.14.24 Reference Temperature Global Variable

Scope: Radiative Flux Boundary Condition

`Reference Temperature Global Variable {=|are|is}` *`GlobalVariableName`*

Parameter	Value	Default
GlobalVariableName	`string`	`undefined`

Summary Specify a global variable to be used for reference temperature.

33.14.25 Reference Temperature Subroutine

Scope: Radiative Flux Boundary Condition

`Reference Temperature Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of a user-defined subroutine that is to be used to calculate the reference temperature associated with this boundary condition.

33.14.26 Reference Temperature Temperature Function

Scope: Radiative Flux Boundary Condition

`Reference Temperature Temperature Function {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of the temperature-dependent function that is to be used to calculate the reference temperature associated with this boundary condition.

33.14.27 Reference Temperature Time Function

Scope: Radiative Flux Boundary Condition

`Reference Temperature Time Function {=|are|is}` *`FunctionName`*

Parameter	Value	Default
FunctionName	`string`	`undefined`

Summary Specify the name of a time-dependent function for the reference temperature for this boundary condition.

33.14.28 Use Banded Wavelength Model

Scope: Radiative Flux Boundary Condition

`Use Banded Wavelength Model` *`ModelName`*

Parameter	Value	Default
ModelName	`string`	`undefined`

Summary Use a specified banded wavelength model for use in enclosure radiation.

Description Requests that a specific banded wavelength model named modelName, be associated with an enclosure, where the modelName is defined outside of the enclosure definition block. Only one BANDED WAVELENGTH MODEL can be used for an enclosure.

33.14.29 Use Bulk Element

Scope: Radiative Flux Boundary Condition

`Use Bulk Element` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Use the named bulk element to model the reference temperature.

Description This line command specifies the name of a bulk fluid element that has been defined using the bulk fluid command block. The temperature of the bulk fluid element is used as the reference temperature, T_r, for the convective heat transfer. Note that it is illegal to specify both a reference temperature and a bulk element.

33.14.30 Use Data Block

Scope: Radiative Flux Boundary Condition

`Use Data Block` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

33.14.31 Use Toggle Block

Scope: Radiative Flux Boundary Condition

`Use Toggle Block` *`ToggleName`* `[ {@|at|for|in|on|over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.15 User Variable

Scope: Equation System

```
Begin User Variable Name

    Add Part part...

    Get Global From Region other_region RemoteVariable Index

    Global Operator {=|are|is} UserVarOperator

    Initial Value {=|are|is} Value...

    Read Variable {=|are|is} Variable [ At Time TimeValue ]

    Reset On Assign

    Type {=|are|is} UserVarMeshObject UserVarType [ Length {=|are|is} Length ]

    Use With Restart

End
```

Summary — Defines user-named registered variables for this region. The name of the variable is specified by the line parameter 'name'.

Description — This command block allows the user to define their own variables, which have the same status as pre-existing variables. For example, they may be output in the same way as other Aria variables, and may participate in SIERRA restart services. This feature is useful when a user has a need to store data on each mesh object (e.g. nodes) or wishes to evaluate a scalar quantity that requires parallel reduction across processors (e.g. a sum). Typically, a user variable would be set by a SIERRA framework transfer service from another region, or via user subroutines.

33.15.1 Add Part

Scope: User Variable

```
Add Part part...
```

Parameter	Value	Default
part	`string...`	`undefined`

Summary — Adds surface, block or mesh group, by name, to a user variable's extent.

Description — This line command is used to add surfaces or blocks to the extent of a user variable. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Similarly element blocks also have a global identifier. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.15.2 Get Global From Region

Scope: User Variable

`Get Global From Region` *`other_region RemoteVariable Index`*

Parameter	Value	Default
other_region	`string`	`undefined`
RemoteVariable	`string`	`undefined`
Index	`integer`	`undefined`

Summary — Specifies the user global variable value will be assigned the value of the named RemoteVariable retrieved from another application Region. Index is an offset into the RemoteVariable.

33.15.3 Global Operator

Scope: User Variable

`Global Operator {= | are | is}` *`UserVarOperator`*

Parameter	Value	Default
UserVarOperator	`{max	min	sum}`	`undefined`

Summary — REQUIRED for GLOBAL variables. Since global variables are often used in user subroutines a need arises in parallel simulations to resolve a representative single GLOBAL variable value across processors. The choice of operator determines how the single global value is to be resolved from each of the individual processor local values of the variable.

For SUM the local global variable values on each processor are summed to obtain a single value of the global variable.

For MIN the local global variable values on each processor are compared and the minimum value over all processors is assigned to the global variable.

For MAX the local global variable values on each processor are compared and the maximum value over all processors is assigned to the global variable.

Parallel synchronization of the single global variables will occur both at the beginning of a nonlinear solution iteration and at the completion of a nonlinear step. Once the synchronization has occurred the local global variable is reset to its initial value.

33.15.4 Initial Value

Scope: User Variable

`Initial Value {= | are | is}` *`Value...`*

Parameter	Value	Default
Value	`real...`	`undefined`

Summary — Optional. Specifies the initial value of the user variable. The number of initial values specified with this line command must match the length of the variable specified in the "TYPE" command. If this command is present the specified initial value will automatically be applied to the user variable.

33.15.5 Read Variable

Scope: User Variable

```
Read Variable {= | are | is} Variable [ At Time TimeValue ]
```

Parameter	Value	Default
Variable	string	undefined

Summary Requests that the variable will be assigned from a time plane of the input ExodusII database. If an AT TIME is not specified, the time plane defaults to the last time plane in the database.

33.15.6 Reset On

Scope: User Variable

Summary Specifies when the user global variable will be reset. By default global variable will be reset after the nonlinear step has converged.

33.15.7 Type

Scope: User Variable

```
Type {= | are | is} UserVarMeshObject UserVarType [ Length {= | are | is} Length ]
```

Parameter	Value	Default
UserVarMeshObject	{edge | element | face | global | node}	undefined
UserVarType	{integer | real | real_tensor | real_vector}	undefined

---

Summary This line command is required. The type of user variable is either real or integer, and is either a single, global value, or is associated with each mesh object of the specified kind. Global variables must have a 'GLOBAL OPERATOR' command associated with them to define its synchronization. The default length is 1. The default initial value for global variables is 0.

33.15.8 Use With Restart

Scope: User Variable

Summary Optional. The default behavior is for user-defined variables to not be preserved across a restart. The presence of this line command specifies that the variable is to be written to and restored from the restart file.

33.16 Volume Heating

Scope: Equation System

```
Begin Volume Heating Name

    Add Volume VolumeList...
```

```
    Chemeq {=|are|is} Model
    Element Subroutine {=|are|is} Name
    Element Variable {=|are|is} Name
    Encore Function {=|are|is} Function
    Field Scaling {=|are|is} Scale_factor
    Integer Data Values...
    Integrated Power Output VariableName
    Joule Heating
    Nodal Variable {=|are|is} Name
    Real Data Values...
    Scaling Time Function {=|are|is} Function
    Scaling With Global Variable {=|are|is} name
    Source Fortran Subroutine {=|are|is} Name
    Temperature Function {=|are|is} Function
    Time Function {=|are|is} Function
    Use Data Block Name
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList... ]
    Use Verdi
    Value {=|are|is} vhs
    Verdi Qdot Scaling {=|are|is} Scale_factor For Keyval...
End
```

Summary Defines a volumetric heat source. Exactly one heating type specification is allowed.

Description The volumetric heating must be applied to at least one volume. If a user subroutine is specified, then real and integer data may be declared that will be local in scope to this instance of the volume heating.

33.16.1 Add Volume

Scope: Volume Heating

```
Add Volume VolumeList...
```

Parameter	Value	Default
VolumeList	string...	undefined

Summary Specifies the mesh volume to which this code feature is to be applied. More than one volume name may be specified, as a space-delimited list.

33.16.2 Chemeq

Scope: Volume Heating

`Chemeq {=|are|is}` *`Model`*

Parameter	Value	Default
Model	`string`	`undefined`

Summary Specifies that volume heating is due to chemical reaction. Here MODEL pertains to the paired CHEMEQ MODEL and CHEMEQ SOLVER command blocks that define the chemical system and how it can evolve. Chemical heating source contributions are further controlled through parameters in the CHEMEQ SOLVER command block.

33.16.3 Element Subroutine

Scope: Volume Heating

`Element Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies that the named user-defined subroutine be used. At most one subroutine name may be specified for a given quantity. This subroutine must conform to the "element signature" argument list type.

33.16.4 Element Variable

Scope: Volume Heating

`Element Variable {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies the name of an element variable in which the volumetric heat source is stored. The units of this term are power per unit volume. Currently, this must be constant in space over each finite element.

33.16.5 Encore Function

Scope: Volume Heating

`Encore Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies that volumetric heating be applied with the named Encore function.

33.16.6 Field Scaling

Scope: Volume Heating

`Field Scaling {=|are|is}` *`Scale_factor`*

Parameter	Value	Default
Scale_factor	`real`	`undefined`

Summary This command is only relevant when the NODAL VARIABLE or ELEMENT VARIABLE command line option is present. The command enables scaling of the Field variable for the element blocks within this Volume Heat command block.

33.16.7 Integer Data

Scope: Volume Heating

`Integer Data` *`Values...`*

Parameter	Value	Default
Values	`integer...`	`undefined`

Summary List of integer data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.16.8 Integrated Power Output

Scope: Volume Heating

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	`string`	`undefined`

Summary Calculate the total power associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.16.9 Joule Heating

Scope: Volume Heating

Summary Specifies that Joule heating will be applied. This model requires that either a VOLTAGE or CURRENT Equation be defined on the specified element block(s). Additionally the corresponding electrical conductivity must be defined.

33.16.10 Nodal Variable

Scope: Volume Heating

`Nodal Variable {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary — Specifies the name of a nodal variable in which the volumetric heat source is stored. The units of this term are power per unit volume.

Description — The nodal variable could be user defined, which could originate from a transfer in the case of coupled mechanics, or from some user subroutine. The nodal values are interpolated to the Gauss points and then integrated over each element.

33.16.11 Real Data

Scope: Volume Heating

`Real Data` *`Values...`*

Parameter	Value	Default
Values	`real...`	`undefined`

Summary — List of real data values to be used by the FORTRAN user subroutine. Copies of these values are provided to the subroutine hence changes to these values within the subroutine are not saved.

33.16.12 Scaling Time Function

Scope: Volume Heating

`Scaling Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary — This command is only relevant when the NODAL VARIABLE or ELEMENT VARIABLE command line option is present. This command enables time function scaling of the field variable for the element blocks within this Volume Heat command block.

33.16.13 Scaling With Global Variable

Scope: Volume Heating

`Scaling With Global Variable {=|are|is}` *`name`*

Parameter	Value	Default
name	`string`	`undefined`

Summary This command is only relevant when the NODE VARIABLE or ELEMENT VARIABLE command line option is present. The command enables scaling of the source Field by a global variable for the surfaces within the Volume Heat command block. The scalar global variable itself must be defined at the Region scope as type Real with length 1.

33.16.14 Source Fortran Subroutine

Scope: Volume Heating

`Source Fortran Subroutine {=|are|is}` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specify the name of a FORTRAN user subroutine that will be used to calculate the volumetric source.

33.16.15 Temperature Function

Scope: Volume Heating

`Temperature Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies a temperature dependent function name for a volumetric heat source to be applied in the named volume.

33.16.16 Time Function

Scope: Volume Heating

`Time Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies a time function dependent name for a volumetric heat source to be applied in the named volume.

33.16.17 Use Data Block

Scope: Volume Heating

`Use Data Block` *`Name`*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Reference to predefined data to be used by the user subroutine. These values may be changed by the user subroutine.

33.16.18 Use Toggle Block

Scope: Volume Heating

`Use Toggle Block` *`ToggleName`* `[ {@ | at | for | in | on | over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary — Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.16.19 Use Verdi

Scope: Volume Heating

Summary — Specifies that the external Goodyear code VERDI should be used to calculate the volumetric heating source term. The value is assumed to be constant in space over each finite element. WARNINGS: There are some caveats: the code will abort (1) with multi-processor runs, since Verdi is inherently serial; (2) with 3-d simulations, since Verdi expects a 2D heat problem to be solved; (3) whenever mesh adaptivity is in use, since Verdi uses three different meshes in its work that must remain very closely related; and (4) whenever Verdi is called from an executable other than eagle (including Calore), since only eagle links against Verdi.

33.16.20 Value

Scope: Volume Heating

`Value {= | are | is}` *`vhs`*

Parameter	Value	Default
vhs	`real`	`undefined`

Summary — Specifies a constant value for a volumetric heat source to be applied in the named volume.

33.16.21 Verdi Qdot Scaling

Scope: Volume Heating

`Verdi Qdot Scaling {= | are | is}` *`Scale_factor`* `For` *`Keyval...`*

Parameter	Value	Default
Scale_factor	`real`	`undefined`
Keyval	`string...`	`undefined`

Summary — This command is only relevant when the USE VERDI command line option is present. The command enables scaling of the element Qdot scale factor for a collection of element blocks. This command can appear within the command block several times with different scaling factors for each set of elements. Qdot scaling for element blocks not appearing in a VERDI QDOT SCALING command line is set to unity.

33.17 Laser Heating

Scope: Equation System

```
Begin Laser Heating Name

    Absorption Coefficient {=|are|is} value
    Absorption Coefficient Encore Function {=|are|is} Function
    Activation Temperature {=|are|is} Value
    Add Volume VolumeList...
    Aft Semi Axis {=|are|is} aft_semiaxis
    Beam Diameter {=|are|is} laser_diameter
    Depth Direction {=|are|is} Function
    Depth Semi Axis {=|are|is} depth_semiaxis
    Distribution {=|are|is} LaserSourceDistributionType
    Effective Beam Radius {=|are|is} value
    Efficiency {=|are|is} Efficiency
    Integrated Power Output VariableName
    Path Function {=|are|is} Function
    Power {=|are|is} laser_power
    Power Encore Function {=|are|is} Function
    Power Time Function {=|are|is} Function
    Source Type {=|are|is} LaserSourceType
    Spatial Influence Factor {=|are|is} Value
    Speed {=|are|is} EquationName
    Start Time {=|are|is} Value
    Stop Time {=|are|is} Value
    Travel Semi Axis {=|are|is} travel_semiaxis
    Use Toggle Block ToggleName [ {@|at|for|in|on|over} ElementBlockList...  ]
    Width Semi Axis {=|are|is} width_semiaxis

End
```

Summary — Defines a moving laser volumetric heat source. Exactly one heating type specification is allowed.

Description — The laser volumetric heating must be applied to at least one volume. If a user subroutine is specified, then real and integer data may be declared that will be local in scope to this instance of the volume heating.

33.17.1 Absorption Coefficient

Scope: Laser Heating

`Absorption Coefficient {=|are|is}` *`value`*

Parameter	Value	Default
value	`real`	`undefined`

Summary Specifies the surface absorption coefficient.

33.17.2 Absorption Coefficient Encore Function

Scope: Laser Heating

`Absorption Coefficient Encore Function {=|are|is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies that absorption coefficient be applied with the named Encore function.

33.17.3 Activation Temperature

Scope: Laser Heating

`Activation Temperature {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`Real_Max`

Summary Defines temperature at which an element may be activated.

Description Elements within a geometric region bounded using SPATIAL INFLUENCE FACTOR will be activated when this temperature is exceeded. Used in conjunction with ACTIVATION_HEMISPHERE or ACTIVATION_SPHERES in addition to the ACTIVATION_USER_FUNCTION thermal conductivity model.

33.17.4 Add Volume

Scope: Laser Heating

`Add Volume` *`VolumeList...`*

Parameter	Value	Default
VolumeList	`string...`	`undefined`

Summary Specifies the mesh volume to which this code feature is to be applied. More than one volume name may be specified, as a space-delimited list.

33.17.5 Aft Semi Axis

Scope: Laser Heating

`Aft Semi Axis {=|are|is}` *aft_semiaxis*

Parameter	Value	Default
aft_semiaxis	`real`	`undefined`

Summary Defines the aft direction semiaxis for the double ellipsoidal and double ellipsoidal laser source model.

33.17.6 Beam Diameter

Scope: Laser Heating

`Beam Diameter {=|are|is}` *laser_diameter*

Parameter	Value	Default
laser_diameter	`real`	`undefined`

Summary Specifies laser spot size.

33.17.7 Depth Direction

Scope: Laser Heating

`Depth Direction {=|are|is}` *Function*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies the depth coordinate direction, X, Y, or Z, the directional axis in which laser is applied. For the ellipsoidal source model may select an additional option, FROM_PATH_NORMAL in which case the depth direction is defined in the opposite direction of the path normal defined by NX, NY, NZ components of the PATH_FUNCTION.

33.17.8 Depth Semi Axis

Scope: Laser Heating

`Depth Semi Axis {=|are|is}` *depth_semiaxis*

Parameter	Value	Default
depth_semiaxis	`real`	`undefined`

Summary Defines the depth direction semiaxis for the ellipsoidal and double ellipsoidal source model.

33.17.9 Distribution

Scope: Laser Heating

`Distribution {=|are|is}` *`LaserSourceDistributionType`*

Parameter	Value	Default
LaserSourceDistributionType	{gaussian|uniform}	GAUSSIAN

Summary Specifies the spatial distribution of applied laser energy.

Description Defines how the laser energy is deposited onto the source geometry. Generally speaking the spatial distribution is GAUSSIAN centered on the source point but for fiber bundles one may opt for the top hat or UNIFORM distribution.

33.17.10 Effective Beam Radius

Scope: Laser Heating

`Effective Beam Radius {=|are|is}` *`value`*

Parameter	Value	Default
value	real	undefined

Summary Specifies effective radius of laser beam.

33.17.11 Efficiency

Scope: Laser Heating

`Efficiency {=|are|is}` *`Efficiency`*

Parameter	Value	Default
Efficiency	real	undefined

Summary Defines the transmission efficiency of the available laser power.

33.17.12 Integrated Power Output

Scope: Laser Heating

`Integrated Power Output` *`VariableName`*

Parameter	Value	Default
VariableName	string	undefined

Summary Calculate the total power associated with this flux boundary condition.

Description This line command specifies that, as a postprocess, the normal flux associated with this boundary condition be integrated over the surface to obtain the total power which is then stored into a global variable named "variableName". This global variable may then be output to history files, or accessed in user subroutines, etc.

33.17.13 Path Function

Scope: Laser Heating

`Path Function {= | are | is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies a time and position function for which a laser source is applied in the named volume.

33.17.14 Power

Scope: Laser Heating

`Power {= | are | is}` *`laser_power`*

Parameter	Value	Default
laser_power	`real`	`undefined`

Summary Specifies strength of the laser source.

33.17.15 Power Encore Function

Scope: Laser Heating

`Power Encore Function {= | are | is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies that power be applied with the named Encore function in the named volume.

33.17.16 Power Time Function

Scope: Laser Heating

`Power Time Function {= | are | is}` *`Function`*

Parameter	Value	Default
Function	`string`	`undefined`

Summary Specifies that power be applied as a function of time in the named volume.

33.17.17 Source Type

Scope: Laser Heating

`Source Type {= | are | is}` *`LaserSourceType`*

Parameter	Value	Default
LaserSourceType	{activation_hemisphere | activation_sphere | beer_lambert | ellipsoidal | hemisphere | integrated_activation_hemisphere}	HEMISPHERE

Summary Selects the laser source model applied to a portion of the underlying volume as defined by a geometric function. When used with a Gaussian DISTRIBUTION all source types produce maximum intensity at the center of the laser spot. Aside from the ACTIVATION SPHERE model these sources are currently limited to paths aligned with the x,y,z coordinate directions.

33.17.18 Spatial Influence Factor

Scope: Laser Heating

Spatial Influence Factor {= | are | is} *Value*

Parameter	Value	Default
Value	real	0.0

Summary Defines a distance from beam focus for which an element may be activated.

Description A multiplier which expands the bounding source geometry. When used with ACTIVATION TEMPERATURE and ACTIVATION_USER_FUNCTION thermal conductivity elements within bounded region become candidates for temperature activation. Used in conjunction with ACTIVATION_HEMISPHERE or ACTIVATION_SPHERE source models.

33.17.19 Speed

Scope: Laser Heating

Speed {= | are | is} *EquationName*

Parameter	Value	Default
EquationName	string	undefined

Summary Specifies the rate at which the laser beam moves across the body.

Description The speed at which the beam moves in traversing the specified geometric path. When a geometric path function is provided the speed is internally computed based upon the START TIME and STOP TIME.

33.17.20 Start Time

Scope: Laser Heating

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

Parameter	Value	Default
Value	real	undefined

Summary Specifies the time at which the laser beam is activated.

33.17.21 Stop Time

Scope: Laser Heating

`Stop Time {=|are|is}` *`Value`*

Parameter	Value	Default
Value	`real`	`undefined`

Summary Specifies the time at which the laser beam is deactivated.

33.17.22 Travel Semi Axis

Scope: Laser Heating

`Travel Semi Axis {=|are|is}` *`travel_semiaxis`*

Parameter	Value	Default
travel_semiaxis	`real`	`undefined`

Summary Defines the travel direction semiaxis for the ellipsoidal and double ellipsoidal source model.

33.17.23 Use Toggle Block

Scope: Laser Heating

`Use Toggle Block` *`ToggleName`* `[ {@|at|for|in|on|over}` *`ElementBlockList...`* `]`

Parameter	Value	Default
ToggleName	`string`	`undefined`

Summary Specification for toggling entities in the computational model based on Toggle Block parameters. When used at the region level, the list of element blocks to be toggled must be provided. Otherwise a listing of entities is not needed as the Toggle Block will be associated with the command line or the enclosing command block.

33.17.24 Width Semi Axis

Scope: Laser Heating

`Width Semi Axis {=|are|is}` *`width_semiaxis`*

Parameter	Value	Default
width_semiaxis	`real`	`undefined`

Summary Defines the width direction semiaxis for the ellipsoidal and double ellipsoidal laser source model.

33.18 Closed Surface Volume

Scope: Equation System

```
Begin Closed Surface Volume ModelName

    Add Surface SurfaceList...

    Utility Group {= | are | is} Frequency

End
```

Summary This command block defines the parameters for computation of the volume of a closed surface. The computed volume is available as a global variable with name *modelName*.

Description Defines a means for computing the volume of a closed surface and associate it with name *modelName*.

33.18.1 Add Surface

Scope: Closed Surface Volume

`Add Surface` *SurfaceList...*

Parameter	Value	Default
SurfaceList	`string...`	`undefined`

Summary Adds surfaces by name, (surface_*id*), to a mesh extent list.

Description This line command is used to add surfaces to a mesh extent list. In Exodus II, surfaces are specified as side sets, that have a global integer identifier. For example, side set 12 would be added by this line command using the surface name surface_12. Note that in SIERRA, each element of an array of strings must be separated by whitespace.

33.18.2 Utility Group

Scope: Closed Surface Volume

`Utility Group {= | are | is}` *Frequency*

Parameter	Value	Default
Frequency	`{begin_nonlinear_solve | end_nonlinear_solve | manually_run | post_accept_solution | post_iterate | post_linear_solve | post_nonlinear_solve | pre_iterate | pre_linear_solve | pre_nonlinear_solve | run_initially | run_once | run_post_linear_system_initialization | run_post_mesh_mod}`	`undefined`

Summary Sets the point in the execution to evaluate utility

33.19 Real and Integer Data Access - Data Block

Aria provides a means by which a user can define problem dependent real and integer data to be used by user subroutines. General commands for usage of this capability are described in the chapter on Data Block 7. For users of the Calore code, these data blocks occupy the role formerly served by REAL DATA and INTEGER DATA commands in that code while providing some additional flexibility. Data Blocks can used both within Calore-style boundary condition, initial condition, volume heat command blocks as well as in Calore-style user subroutines.

Data Block information can be accessed by Calore-style subroutines in two ways. The information can be directly accessed through user query functions referencing the Data Block by name within a user subroutine or referencing indirectly by implicit association with a command block, again through user queries.

In either event, data extraction within the user subroutine is accomplished in the same manner but using different interfaces. Accessing Data Block values from a user subroutine involves a simple three-step extraction sequence

1. Define a variable of data type associated with the values one wishes to extract from the Data Block.
2. Define a string with the name of the values to be extracted.
3. Perform a user_query request to retrieve the data from the Data Block.

For FORTRAN-like C subroutines the data type translates to RealArray1d, RealArray2d, IntArray1d or IntArray2d. Usage of this information within a user subroutine is best described using simple examples that use single values from the data block. What remains here is to describe the various user_query requests of data block information for FORTRAN-like C subroutines. Use of Data Block in FORTRAN subroutines is discussed in Section 33.26.

33.19.1 Data Block Referenced From Command Block

In order to reference a Data Block from a command block the Data Block must first be bound to the scope of the command block by including the *USE DATA BLOCK* directive within the scope of the command block. The choice of scope will dictate the type of user query function used to access the Data Block information.

Data Block Referenced From Region Command Block

When the Data Block is defined within the Region scope this information can be made available to the user subroutine system by supplying the *USE DATA BLOCK* directive at the Region scope. From the user subroutine, this data can then be retrieved for usage without knowledge of the data block name. Let the input file contain a data block definition within the Region scope

```
Begin Procedure my_procedure
  .
  begin data block flux_data
    real flux = -0.05
  end data block flux_data
  .
  Begin Aria Region myregion
   .
   use data block flux_data
```

```
    .
    Begin heat flux boundary condition freddy
      add surface surface_4
      element subroutine = aucal_heat_flux
    End
    .
   End Aria Region

  End Procedure
```

Within the user subroutine source code the values can then be retrieved from the Data Block by following the extraction sequence targeting Region data

```
RealArray1d real_array;
sierra::String flux_label("flux");

user_query.getUserRealRegionData(real_array, flux_label);

Real flux_value = real_array(0);
```

Here we note the presence of *Region* in the user_query and the user subroutine becomes

```
int
aucal_heat_flux(
UserQuery &                          user_query,
ElementIds                           element_ids,
Int                                  num_elements,
Int                                  num_integration_points,
Int                                  spatial_dimension,
CoordinateElementIntegrationPoints   coordinates,
TemperatureElementIntegrationPoints  temperature,
ScalarElementIntegrationPoints       scalar)
{
  RealArray1d real_array;
  sierra::String flux_label("flux");

  user_query.getUserRealRegionData(real_array, flux_label);

  Real flux_value = real_array(0);

  for (int q = 0; q < num_integration_points; ++q)
  {
    for (int i = 0; i < num_elements; ++i)
    {
      scalar(i, q) = flux_value;
    }
  }
  return 0;
}
```

Here we note that the Data Block name, *flux_data*, appeared only at the Region scope of the input.

Data Block Referenced From BC, IC or Volume Source Command Block

Data Block information can be made accessible to boundary condition, initial condition or volumetric source user subroutines defined within command blocks. This is accomplished by simply supplying the *USE DATA BLOCK* directive at the command block scope. Returning to the example, the *USE DATA BLOCK* enables the association of *flux_data* with the *aucal_heat_flux* user subroutine.

```
Begin Procedure my_procedure
  .
  begin data block flux_data
    real flux = -0.05
  end data block flux_data
  .
  Begin Aria Region myregion
   .
   use data block flux_data
   .
   Begin heat flux boundary condition freddy
     add surface surface_4
     element subroutine = aucal_heat_flux
     use data block flux_data
   End
   .
  End Aria Region

End Procedure
```

Within the user subroutine source code the values can then be retrieved from the Data Block by following the extraction sequence

```
RealArray1d real_array;
sierra::String flux_label("flux");

user_query.getUserRealInstanceData(real_array, flux_label);

Real flux_value = real_array(0);
```

Here we note the presence of *Instance* in the user_query and the user subroutine becomes

```
int
aucal_heat_flux(
  UserQuery &                        user_query,
  ElementIds                         element_ids,
  Int                                num_elements,
  Int                                num_integration_points,
  Int                                spatial_dimension,
  CoordinateElementIntegrationPoints  coordinates,
  TemperatureElementIntegrationPoints temperature,
  ScalarElementIntegrationPoints      scalar)
{
  RealArray1d real_array;
  sierra::String flux_label("flux");
```

```
  user_query.getUserRealInstanceData(real_array, flux_label);

  Real flux_value = real_array(0);

  for (int q = 0; q < num_integration_points; ++q)
  {
    for (int i = 0; i < num_elements; ++i)
    {
      scalar(i, q) = flux_value;
    }
  }
  return 0;
}
```

Data Block Referenced From Material Command Block

The Data Block information can be made automatically available for material property evaluations employing user subroutines.

```
  Begin Procedure my_procedure
    .
    begin data block k_data
      real my_K = 4.0
    end data block k_data

    begin Aria material my_material
      .
      use data block flux_data
      thermal conductivity = calore_user_sub name = aucal_elem_cond type=element
      .
    end Aria material my_material
    .
    Begin Aria Region myregion
     .
     .
     .
    End Aria Region

  End Procedure
```

Within the user subroutine source code the values can then be retrieved from the Data Block by following the extraction sequence

```
  RealArray1d real_array;
  sierra::String k_label("my_k");

  user_query.getUserRealMaterialData(real_array, k_label);

  Real k_value = real_array(0);
```

Here we note the presence of *Material* in the user_query and the user subroutine becomes

```
int
aucal_elem_cond(
  UserQuery & user_query,
  ElementIds element_ids,
  Int num_elements,
  Int num_integration_points,
  Int spatial_dimension,
  CoordinateElementIntegrationPoints coordinates,
  TemperatureElementIntegrationPoints temperature,
  ScalarElementIntegrationPoints cond)
{
  RealArray1d real_array;
  sierra::String k_label("my_k");

  user_query.getUserRealMaterialData(real_array, k_label);

  Real k_value = real_array(0);

  for (int q = 0; q < num_integration_points; ++q)
  {
    for (int i = 0; i < num_elements; ++i)
    {
      cond(i, q) = k_value(i);
    }
  }
  return 0;
}
```

The previous examples demonstrated association of Data Block information with the scope of a command block in order to avoid explicit specification of the Data Block name. However, alternative access methods also allow one to retrieve Data Block information more directly by explicitly supplying the Data Block name in the user_query operation rather than by supplying the *USE DATA BLOCK* directive at a given scope. A complete list of the various query operations is included in the section on For the examples previously described this style of querys translate to

```
  user_query.getUserRealData(real_array, flux_label,"flux_data");
 and
  user_query.getUserRealData(real_array, k_label,"k_data");
```

For the simple examples provided the type of user_query is a personal choice. However, an overall strategy for usage of Data Block information within user subroutines often dictates the type of user_query employed. Considerations in this strategy often include the number of subroutines that will be supported and how the subroutines are managed. As an example one could configure all input files in the same way and selectively load subroutines that employ the same interface. Again we emphasize that the different methods of accessing Data Block information differ only by the interface used to obtain the data definition.

33.19.2 Data Block Referenced By Line Command

For commands not defined within a Command Block the Data Block can still be used to specify parameters for that line command using the *USE DATA BLOCK* directive. The data block parameters are placed in the line command as if it was a command block.

Creating the data block in the region using the data block command block:

```
begin data block bc_flux
  real h = 300.0
  integer t_ref = 1
end
```

its data can be accessed using a command line like

```
BC Flux for Energy on no_slip_boundary = Nat_Conv use data block bc_flux
```

which is equivalent to

```
BC Flux for Energy on no_slip_boundary = Nat_Conv H=300.0  T_ref=1
```

This architecture allows the data to be consistently used with boundary conditions for several equations.

33.20 User Subroutines and Variables

Aria offers support of C language Calore-style user subroutines and FORTRAN user subroutines through fixed-interface signatures. Callback data query requests from the subroutines are also supported, again through fixed-interface function signatures. It is important to note that Calore-style subroutines provide some level of variable-type safety and bounds checking that is not supported with FORTRAN subroutines.

Some of the subroutine interface callback function calls enable writing to user defined node, element or face Fields, which can be used internal to the code or output to the results file. Generally speaking, values made available to a subroutine through its calling interface are quadrature point values. Internally the application code can either process all quadrature points at one time or all at once. Hence in order to save user fields that are populated within the subroutine one should understand exactly how the subroutine is being processed before relying upon data saved from within the subroutine.

Usage of both C language Calore-style 33.21 and FORTRAN subroutines 33.25 within Aria are described in the sections which follow.

33.21 C Calore-Style User Subroutines and Variables

Aria supports the use of C language user subroutines. Historically, the Calore heat transfer code supported only FORTRAN subroutines. In later releases of Calore, support of FORTRAN subroutines has continued but now C language style user functions are supported as well. With only minor syntactical changes these C language user functions support the full functionality of FORTRAN subroutines. When written in a general manner, the C Calore-style user functions can be used in either Calore or Aria. The overall nature of the Calore style user function is described below and is followed by a simple example of its use with the DATA_BLOCK utility. In what follows, the terms "signature", "argument list", and "interface" are synonyms for the list of variables and arrays that are passed into a user subroutine. Furthermore, the term Calore style user subroutine will be loosely associated with the term Calore style user functions.

An Aria Calore-style user subroutine must be a text file of suffix .C. In contrast to the static link approach formerly used in the Calore code, here the file contents are made available to the application as a dynamically linked function using the same procedure as for Aria User Plugin functions 18. For example, a file named Aucal_density.C the file will be referenced from the Aria input file using the line command

```
load user plugin file ./Aucal_Density.so
```

Note that, since Aucal_density.C is a filename, its input file name is case-sensitive, and considerations must be made concerning its location path. The user is free to name the file and subroutines in any way, provided the name is consistent with compiler and operating system limitations. For example, special characters such as '*' and ',' should be avoided. Finally actual use of the function by the application is triggered by association of the "model name", calore_user_sub and the user subroutine name in particular command line, i.e. initial condition, source or boundary condition.

Multiple user subroutines for a given parameter, e.g. conductivity, may be defined. If user conductivity for two materials, say gold and silver was needed then one could write a subroutine for each of the two conductivities. Each subroutine must have a different name in order to be properly referenced, and each must conform to the required call signature. This approach eliminates the need for "if-then-else if" subroutine constructs defining specific behavior for a particular material, volume or surface.

Regardless of the information included in the call signature, the need will arise to access data that is not currently provided by the subroutine arguments. Altering the call signature to fulfill that need would require extensive changes to Calore style user subroutines. Furthermore, backward compatibility between code versions would be lost, since all subroutines currently using an older interface would no longer function properly with a newer interface.

For reasons previously stated, the design philosophy behind the user subroutine capability in both Calore and Aria is that the call signature will kept to a minimum. An advantage of this approach is that it provides a clean, simple interface. A disadvantage is that it limits the information that a user may access. This inherent disadvantage is addressed by employing query functions for use within the subroutines. Thus if one wishes to access additional information from the user subroutine, it becomes a relatively straightforward task to implement a new query function to provides the required data. In this way, no changes are required of the subroutine signature, and backward compatibility is maintained. Users having specialized needs of data access should consider implementations of native Aria user plugin functions instead [18].

An attempt has been made to ease the transition from Calore user subroutines to Calore style user subroutines in Aria by providing common C interfaces to the two codes. Two versions of the subroutine signatures are provided here, the first more reminiscent of a FORTRAN interface and the other being a C language interface.

In many applications several user subroutines are being used in a single simulation. In this regard it is worth mentioning that with both the FORTRAN and C-style usage one must register the various subroutines/functions with the application before they can be recognized and accessed by the application. This registration establishes an association between the a code interface, function signature and the name by which the function will be referenced from the input file command lines. For the Aria C-style subroutines the registration can be done in one of two ways, with individual registration lines for each subroutine or by using a single function to register one or more subroutines.

Subroutine registration amounts to adding a line to the subroutine source file with a line declaring the signature corresponding to the subroutine containing its registered name and a pointer to the user subroutine. As an example, if a user subroutine file named Aucal_Density.C contained a function named my_density and one wished to reference the function as bulk_density one could register the subroutine as:

```
RegisterFunction(ElemUserSubC, my_density, "bulk_density");
```

Alternatively one could also use a more obscure form of the registration

```
sierra::Aria::AcalElemUserSubC<&my_density>::Register distrr_register("bulk_density");
```

Again we note that both registration methods contain the application interface signature, the function

pointer and the name one wishes to associate the function with from the input file. This registration causes the subroutine to be recognized within the function name registry. Since the compiled function name is not available prior to compilation, the registration line should appear after the subroutine function in the subroutine source code.

Access to the subroutine from within the application is enabled using the following command line:

```
load user plugin file ./Aucal_Density.so
```

Subroutine registration using a single function can be performed simply by wrapping the individual registrations previously described in a function. Using the subroutine name previously discussed the corresponding registration using a function call could be obtained using the following module

```
extern "C"
void
bulk_prop_subs()
{
  RegisterFunction(ElemUserSubC, my_density, "bulk_density");
}
```

As in the case of single line subroutine registration compiled function names are not available prior to compilation, thus the registration function must appear in the subroutine source code after the subroutine function being registered.

Access to the subroutine from within the application is enabled using the following command line:

```
load user plugin file ./Aucal_Density.so using function bulk_prop_subs
```

One can also avoid using a named registration module by using the default function name $dl\_register$,

```
extern "C"
void
dl_register()
{
  RegisterFunction(ElemUserSubC, my_density, "bulk_density");
}
```

where once again access to the subroutine from within the application is enabled without using a function name

```
load user plugin file ./Aucal_Density.so
```

Finally in order to access the subroutine from the input file according to its usage from the Aria material command block one would reference the subroutine as

```
Begin Aria Material bulk
  .
  .
  density = Calore_User_Sub  name=bulk_density type = element
  .
End Aria Material bulk
```

33.22 FORTRAN-like C Interfaces

While Calore style user subroutines could be defined to supply many different parameters, only five different signatures are provided in Aria. These signatures characterize the argument lists by which the subroutine interacts with the code. The various subroutine signatures are denoted as:

- node signature 33.22.1
- element signature 33.22.2
- scaled element signature 33.22.3
- reference temperature dependent element signature 33.22.4
- scaled, reference temperature dependent element signature 33.22.5
- tensor thermal conductivity signature 33.22.6
- transformed tensor thermal conductivity signature 33.22.7
- reaction rate signature 33.22.8
- auxiliary variable signature 33.22.9.

The node subroutine signature is generally used for Dirichlet boundary conditions. The element signature is used to define material properties, heat flux and heat transfer coefficients. The reference temperature dependent signatures are used primarily for defining heat transfer coefficients. The last two signatures are specific to the CHEMEQ chemistry package. Scaled signatures are used to combine the basic user subroutine functionality combined with Field variable scaling. The different subroutine call signatures are outlined below. With the exception of the *node* signature, these signatures will be used for subroutines that operate on quadrature point values.

It is worth noting that the purpose of the user subroutine is simply to provide an evaluation of needed values based upon the data supplied through the subroutine interface. In many cases the data supplied to the user subroutine is more that what is necessary for the targeted evaluation. Here the user is free to perform the evaluation using only the information needed to perform the target evaluation. Note that only the last argument of the interface is associated with values returned to the application code, the remaining arguments represent non-modifiable data local to the element, face or node being processed.

33.22.1 Node Signature

For example, to specify a temperature Dirichlet boundary condition, the node signature is

```
/**
 * Prototype for the nodal signature.
 *
 * user_query: query function
 * node_id:    array of global ids of the nodes
 * num_nodes:         number of nodes in the array
 * spatial_dimension: number of nodes in the array
 * coordinate_array: array of coordinates of the current nodes
 * node_data_array:  array of value to be calculated
 */
Int
```

```
AcalNodeUserSubC(
  UserQuery &     user_query,
  Int             node_id,
  Int             num_nodes,
  Int             spatial_dimension,
  CoordinateNodes coordinate_array,
  DataNodes       node_data_array);
```

The node subroutine for a Dirichlet boundary condition could be invoked using a command line:

```
BC Dirichlet for Temperature on surface_4 = Calore_User_Sub name=dirich_bc type=node
```

A simple example using the nodal signature subroutine is as follows:

```
#include <utility>
#include "math.h"

#include "Aria_Calore_User_Sub_Support.h"

int
dirichsub(
  UserQuery &                          user_query,
  Int nodeid,
  Int nnode,
  Int spatial_dimension,
  CoordinateNodes coords,
  DataNodes temperature)
{
  for (int i = 0; i < nnode; ++i) {
    Real ynode = coords(1,i);
    if ( ynode < 0.5 ) {
       temperature(i) = 1.5;
    } else {
       temperature(i) = 0.1;
    }
  }
  return 0;
}

RegisterFunction(NodeUserSubC, dirichsub, "dirich_bc");
```

33.22.2 Element Signature

For density, specific heat, volumetric heat source, surface heat flux and most surface heat transfer coefficients defined via user subroutine one will use an element based call signature. Additionally, for problems with isotropic thermal conductivity one can supply the single value of conductivity using the element signature. In either case the subroutine will be used to assign quadrature point evaluations. That is, the coordinate_array values can be used to compute node_data_array values The call signature for scalar element-based user subroutines is

```
/**
```

```
 * Prototype for the scalar element signature.
 *
 * user_query: query function
 * elem_ids: pointer to the global id of the current element
 * num_elements: pointer to the number of elements in the workset
 * nint: pointer to the number of gauss points
 * coordinate_array: pointer to the coordinates of the current node (3, nelem, nint)
 * temperature_array:  pointer to the temperature (nelem, nint)
 * elem_data_array: pointer to the values to be calculated (nelem, nint)
 */
Int
AcalElemUserSubC(
  UserQuery &                           user_query,
  ElementIds elem_ids,
  Int                                   num_elements,
  Int                                   nint,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coordinate_array,
  TemperatureElementIntegrationPoints   temperature_array,
  ScalarElementIntegrationPoints        element_data_array)
```

The scalar element subroutine signature is supported for all element-centric evaluations. An example of specifying the element user subroutine to be used for density is as follows:

```
  Begin Aria Material sticky_icky
    density = calore_user_sub name = aucal_density type=element
    specific heat = constant cp = 385.0
    thermal conductivity = constant k = 400.0
    heat conduction = basic
End Aria Material sticky_icky
```

A simple example using the scalar element signature subroutine is as follows:

```
#include <utility>
#include "math.h"

#include "Aria_Calore_User_Sub_Support.h"

int
aucal_density(
  UserQuery &                           user_query,
  ElementIds                            elemid,
  Int                                   nelem,
  Int                                   nint,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coords,
  TemperatureElementIntegrationPoints   temp,
  ScalarElementIntegrationPoints        rho)
{
  //  Load the density array
  for (int j = 0; j < nint; ++j) {
    for (int i = 0; i < nelem; ++i) {
      rho(i,j) = 0.1 + temp(i,j);
```

```
    }
  }
  return 0;
}

RegisterFunction(ElemUserSubC, aucal_density, "aucal_density");
```

The same methodology used in the previous example is valid when defining isotropic thermal conductivity that depends upon other local state information. Here all directions are material principal directions so thermal conductivity will be a single value for each quadrature points of an element.

Note that in this example the user subroutine name is specified within quotation marks. As per 2.6.1 the quotation marks are only required when the name by which the subroutine is referenced by from the input file begins with a number. Otherwise quotation marks are note needed in the specification of user subroutine name.

```
  BEGIN ARIA MATERIAL my_mat
    density = constant rho = 0.1
    specific heat = constant cp = 385.0
    thermal conductivity = calore_user_sub name="2Dcond" type = element
    heat conduction = basic
  END ARIA MATERIAL my_mat
```

For isotropic conductivity an example of the associated user subroutine and its function registration would be:

```
#include <utility>
#include "math.h"

#include "Aria_Calore_User_Sub_Support.h"

int
mycond(
  UserQuery & user_query,
  ElementIds element_ids,
  Int num_elements,
  Int num_integration_points,
  Int spatial_dimension,
  CoordinateElementIntegrationPoints coordinates,
  TemperatureElementIntegrationPoints temperature,
  ScalarElementIntegrationPoints cond)
{
  for (int q = 0; q < num_integration_points; ++q)
  {
    for (int i = 0; i < num_elements; ++i)
    {
      cond(i, q) = 400.0;
    }
  }

  return 0;
}
```

```
RegisterFunction(ElemUserSubC, mycond, "2Dcond");
```

33.22.3 Scaled Element Signature

A variant of the element subroutine employs an additional Field variable that is supplied through the user interface. Here the Field variable is used to perform some logical decision on how the result is assigned and the name of the Field variable to be used is specified from the input file. The call signature for scaled scalar element-based user subroutines is

```
/**
 * Prototype for the scaled scalar element signature.
 *
 * user_query: query function
 * elem_ids: pointer to the global id of the current element
 * num_elements: pointer to the number of elements in the workset
 * nint: pointer to the number of gauss points
 * coordinate_array: pointer to the coordinates of the current node (3, nelem, nint)
 * temperature_array:  pointer to the temperature (nelem, nint)
 * scaling_array:  pointer to the scaling array(nelem, nint)
 * elem_data_array: pointer to the values to be calculated (nelem, nint)
 */
Int
AcalScaledElemUserSubC(
  UserQuery &                           user_query,
  ElementIds elem_ids,
  Int                                   num_elements,
  Int                                   nint,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coordinate_array,
  TemperatureElementIntegrationPoints   temperature_array,
  ScalarElementIntegrationPoints        scaling_array,
  ScalarElementIntegrationPoints        element_data_array)
```

The Scaled Element Signature is supported only for all element property evaluations. A corresponding entry in the convective flux boundary condition command block might be:

```
begin convective flux boundary condition convect
  add surface surface_2
  reference temperature = 180.0
  scaled convective coefficient subroutine is aucal_htcoef fieldname = c0
end convective flux boundary condition convect
```

An example of the scaled scalar element signature subroutine is as follows:

```
#include <utility>
#include "math.h"

#include "Aria_Calore_User_Sub_Support.h"

int
```

```
aucal_htcoef(
  UserQuery &                           user_query,
  ElementIds faceid,
  int nelemf,
  int nint,
  int dimension,
  CoordinateElementIntegrationPoints coords,
  TemperatureElementIntegrationPoints t,
  ScalarElementIntegrationPoints scaling,
  ScalarElementIntegrationPoints htc)
{
  for (int j = 0; j < nint; ++j)
  {
    for (int i = 0; i < nelemf; ++i)
    {
      Real x = coords(0,i,j);
      Real y = coords(1,i,j);
      Real r = std::sqrt(x*x+y*y);
      Real val = 0.0;
      Real c0 = scaling(i,j);
      if( c0 > 0.1 ) {
        val = 45.0;
      } else {
        val = 20.0 * r;
      }
      htc(i,j) = val;
    }
  }

  return 0;
}

RegisterFunction(ScaledElemUserSubC, aucal_htcoef, "aucal_htcoef");
```

One should note that the number of arguments for the Scaled Element Signature and Reference Temperature Dependent Element Signature are identical. Attempts to use a user subroutine in the wrong context will be detected while parsing the input file.

33.22.4 Reference Temperature Dependent Element Signature

In some problems the surface heat transfer coefficient is determined to be a function of reference temperature rather than just the surface temperature. One example of this would be when modeling problems of natural convection where one computes a Rayleigh number based upon the difference between surface temperature and reference temperature to determine the local Nusselt number and a local heat transfer coefficient. In these cases it is sometimes more convenient to utilize an existing model for reference temperature and use those values to evaluate the heat transfer coefficient based upon additional criteria within a user subroutine. Here the interface for the Element Signature is modified to also supply interpolated values of the heat transfer coefficient at the quadrature points with the following call signature

```
/**
 * Prototype for the scalar reference temperature dependent
 * heat transfer coefficient signature.
```

```
 *
 * user_query: query function
 * elem_ids: pointer to the global id of the current element
 * num_elements: pointer to the number of elements in the workset
 * nint: pointer to the number of gauss points
 * coordinate_array: pointer to the coordinates of the current node (3, nelem, nint)
 * ref_temperature_array:  pointer to the reference temperature (nelem, nint)
 * temperature_array:  pointer to the temperature (nelem, nint)
 * elem_data_array: pointer to the values to be calculated (nelem, nint)
 */
Int
AcalElemUserSubHtC(
  UserQuery &                           user_query,
  ElementIds elem_ids,
  Int                                   num_elements,
  Int                                   nint,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coordinate_array,
  ScalarElementIntegrationPoints        ref_temperature_array,
  TemperatureElementIntegrationPoints   temperature_array,
  ScalarElementIntegrationPoints        element_data_array)
```

The Reference Temperature Dependent Element Signature is currently supported only for the heat transfer coefficient. A corresponding entry in the convective flux boundary condition command block might be:

```
begin convective flux boundary condition convect
  add surface surface_2
  reference temperature = 180.0
  ref temp convective coefficient subroutine is aucal_htcoef
end convective flux boundary condition convect
```

The corresponding implementation of the reference temperature heat transfer coefficient subroutine is shown below.

```
#include <utility>
#include "math.h"

#include "Aria_Calore_User_Sub_Support.h"

int
aucal_htcoef(
  UserQuery &                           user_query,
  ElementIds faceid,
  int nelemf,
  int nint,
  int dimension,
  CoordinateElementIntegrationPoints coords,
  ScalarElementIntegrationPoints ref_temp,
  TemperatureElementIntegrationPoints temp,
  ScalarElementIntegrationPoints htc)
{
  Real c1 = 1.0+std::pow(std::pow((0.492/.68),(9./16.)),(8./27));
  Real c2 = 30.0e-3/2.2;
```

```
  for (int j = 0; j < nint; ++j)
  {
    for (int i = 0; i < nelemf; ++i)
    {
      Real Ra = 9.0e6*(temp(i,j)-ref_temp(i,j));
      Real Nu = std::pow((0.825+ 0.387*std::pow(Ra,(1./6))/c1),2.0);
      htc(i,j) = c2 * Nu;
    }
  }
  return 0;
}

RegisterFunction(ElemUserSubHtcC, aucal_htcoef, "aucal_htcoef");
```

33.22.5 Scaled Reference Temperature Dependent Element Signature

In cases where the surface heat transfer coefficient is a function of reference temperature the coefficient and one wishes to either scale the heat transfer coefficient or assign it based upon values of another variable, the heat transfer coefficient can be defined via user subroutine using yet another alternative call signature is

```
/**
 * Prototype for the scaled scalar reference temperature dependent
 * heat transfer coefficient signature.
 *
 * user_query: query function
 * elem_ids: pointer to the global id of the current element
 * num_elements: pointer to the number of elements in the workset
 * nint: pointer to the number of gauss points
 * coordinate_array: pointer to the coordinates of the current node (3, nelem, nint)
 * ref_temperature_array:  pointer to the reference temperature (nelem, nint)
 * temperature_array:  pointer to the temperature (nelem, nint)
 * scale_elem_data_array: pointer to the scaling values (nelem, nint)
 * elem_data_array: pointer to the values to be calculated (nelem, nint)
 */
Int
AcalScaledElemUserSubHtcc(
  UserQuery &                           user_query,
  ElementIds element_ids,
  Int                                   num_elements,
  Int                                   num_integration_points,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coordinate_array,
  ScalarElementIntegrationPoints        ref_temperature_array,
  TemperatureElementIntegrationPoints   temperature_array,
  ScalarElementIntegrationPoints        scaling_array,
  ScalarElementIntegrationPoints        flux_array)
```

The Scaled Reference Temperature Dependent Element Signature is currently supported only for the heat transfer coefficient. A corresponding entry in the convective flux boundary condition command block might be:

```
begin convective flux boundary condition convect1
```

```
  add surface surface_1
  reference temperature = 215.0
  scaled ref_temp convective coefficient subroutine = aucal_htcoef fieldname = c0
end convective flux boundary condition convect1
```

A simple example using the scaled reference temperature heat transfer coefficient signature subroutine is as follows:

```
#include <utility>
#include "math.h"

#include "Aria_Calore_User_Sub_Support.h"

int
aucal_htcoef(
  UserQuery &                           user_query,
  ElementIds faceid,
  int nelem,
  int nint,
  int dimension,
  CoordinateElementIntegrationPoints coords,
  ScalarElementIntegrationPoints ref_temp,
  TemperatureElementIntegrationPoints temp,
  ScalarElementIntegrationPoints scaling,
  ScalarElementIntegrationPoints htc)
{
  for (int j = 0; j < nint; ++j)
  {
    for (int i = 0; i < nelem; ++i)
    {
      Real val = 0.0;
      Real c0 = scaling(i,j);
      Real tref = ref_temp(i,j);
      if( c0 > 0.1 ) {
        val = 45.0*tref*c0;
      } else {
        val = 10.0 * tref;
      }
      htc(i,j) = val;
    }
  }

  return 0;
}

RegisterFunction(ScaledElemUserSubHtcC, aucal_htcoef, "aucal_htcoef");
```

33.22.6 Tensor Thermal Conductivity Signature

When the thermal conductivity is anisotropic and and can be easily defined in the Cartesian coordinate frame one can specify the thermal conductivity tensor with a user subroutine. The call signature for anisotropic tensor thermal conductivity user subroutines is

```
/**
 * Prototype for the tensor element conductivity signature.
 *
 * user_query: query function
 * elem_ids: pointer to the global id of the current element
 * num_elements: pointer to the number of elements in the workset
 * nint: pointer to the number of gauss points
 * coordinate_array: pointer to the coordinates of the current node (3, nelem, nint)
 * temperature_array:  pointer to the temperature (nelem, nint)
 * conductivity_array: pointer to the output principal conductivities (3, 3, nelem, nint)
 */
Int
AcalElemTensorUserSubC(
  UserQuery &                           user_query,
  ElementIds elem_ids,
  Int                                   num_elements,
  Int                                   nint,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coordinate_array,
  TemperatureElementIntegrationPoints   temperature_array,
  TensorElementIntegrationPoints        conductivity_array);
```

In order to use this signature to assign the tensor thermal conductivity for a two dimensional problem the subroutine is first specified in the Aria material command block. Note that a tensor conductivity requires a generalized model of heat conduction.

```
Begin Aria Material mat1
  .
  tensor thermal conductivity = calore_user_sub name = aucal_tensor_cond type=element_tensor
  .
  heat conduction = generalized
End Aria Material mat1
```

A sample implementation of the corresponding user subroutine could be as shown below.

```
#include <utility>
#include "math.h"

#include "Aria_Calore_User_Sub_Support.h"

int
aucal_tensor_cond(
  UserQuery & user_query,
  ElementIds elemid,
  Int nelem,
  Int nint,
  Int spatial_dimension,
  CoordinateElementIntegrationPoints coords,
  TemperatureElementIntegrationPoints temp,
  TensorElementIntegrationPoints cond)
{
  cond.fill(0.0);
```

```
  for (int j = 0; j < nint; ++j) {
    for (int i = 0; i < nelem; ++i) {
      cond(0,0,i,j) = 400.0;
      cond(1,1,i,j) = 300.0;
    }
  }
  return 0;
}

RegisterFunction(ElemTensorUserSubC, aucal_tensor_cond, "aucal_tensor_cond");
```

Note that one could have just as easily defined this conductivity from the Aria material command block as:

```
BEGIN ARIA MATERIAL my_mat
  .
  tensor thermal conductivity = constant t11=400.0 t22=300.0
  heat conduction = generalized
END ARIA MATERIAL my_mat
```

33.22.7 Transformed Tensor Thermal Conductivity Signature

When the thermal conductivity is anisotropic only in the reference frame of local coordinates the conductivity tensor must be transformed into the Cartesian reference frame for computation. Here the user subroutine will define the thermal conductivity tensor components and the principal directions associated with those components. The call signature for anisotropic tensor thermal conductivity user subroutines is

```
/**
 * Prototype for the tensor element principal conductivity
 * and principle direction orientation signature.
 *
 * user_query: query function
 * elem_ids: pointer to the global id of the current element
 * num_elements: pointer to the number of elements in the workset
 * nint: pointer to the number of gauss points
 * coordinate_array: pointer to the coordinates of the current node (3, nelem, nint)
 * temperature_array:  pointer to the temperature (nelem, nint)
 * conductivity_array: pointer to the output principal conductivities (3, 3, nelem, nint)
 * principle_direction_array: pointer to the output principal directions (2, 3, nelem, nint)
 */
Int
AcalCondUserSubC(
  UserQuery &                           user_query,
  ElementIds                            elem_ids,
  Int                                   num_elements,
  Int                                   nint,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coordinate_array,
  TemperatureElementIntegrationPoints   temperature_array,
  ConductivityElementIntegrationPoints  conductivity_array,
  DirectionElementIntegrationPoints     principle_direction_array);
```

Usage of the transformed tensor conductivity subroutine is considerably more detailed than other subroutines. Here we consider an example where the conductivity is known within an element and the orientation of principle directions relative to the Cartesian frame is also known.

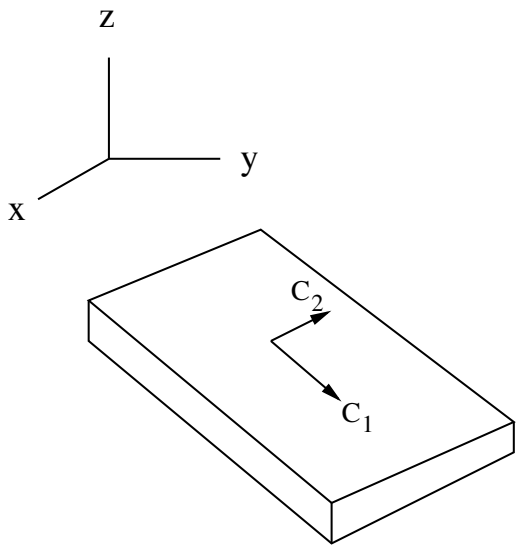


Figure 33.3. Anisotropic conductivity Orientation within Cartesian frame.

Anisotropic Conductivity and Orientation from Input Data:
Note that here the data block is associated with thermal conductivity via the `data = cond_values` argument to thermal conductivity command line.

```
load user plugin file ./aniso.so

Begin data block cond_values
  real cond_data = ( 400.0, 200.0, 400.0, 0.0, 0.0, 0.0 )
  real cond_vec1 = (  0.707107,  0.707107,  0.0 )
  real cond_vec2 = ( -0.707107,  0.707107,  0.0 )
End data block

Begin Aria Material my_material
  density = constant  rho = 0.1
  tensor thermal conductivity = calore_user_sub name = aucal_aniso_cond  \$
     type = element_direction_tensor  data = cond_values
  specific heat = constant cp = 385.0
  heat conduction = generalized
End Aria Material my_material
```

Included below is an example user subroutine implementation of anisotropic thermal conductivity. In this implementation the vectors are indicated as being supplied from the code interface, either from input file data or from the mesh database. Alternatively, one could conditionally define the orientation vector and thermal conductivity within the user subroutine itself. This approach potentially requires selection/identification of each element by spatial location.

Note that each of the modules contains the essential features, orientation vectors, normalization and conductivity values. Once these values are supplied then the transformation operations can be performed internal to the code. This examples also demonstrates the subroutine registration process.

```
Int
aucal_aniso_cond(
  UserQuery &                           user_query,
  ElementIds                            objid,
  Int                                   nelem,
  Int                                   nint,
  Int                                   spatial_dimension,
  CoordinateElementIntegrationPoints    coords,
  TemperatureElementIntegrationPoints   temperature,
  ConductivityElementIntegrationPoints  elemCond,
  DirectionElementIntegrationPoints     elemVec)
{
    RealArray1d cond_data(6);
    RealArray1d cond_vec2(3);
    RealArray1d cond_vec1(3);

    user_query.getUserRealInstanceData(cond_data, "cond_data");
    user_query.getUserRealInstanceData(cond_vec1, "cond_vec1");
    user_query.getUserRealInstanceData(cond_vec2, "cond_vec2");

//     normalize the vectors
    Real rlen = std::sqrt(cond_vec1(0)*cond_vec1(0)
                        +cond_vec1(1)*cond_vec1(1)
                        +cond_vec1(2)*cond_vec1(2));

    cond_vec1(0) = cond_vec1(0) / rlen;
    cond_vec1(1) = cond_vec1(1) / rlen;
    cond_vec1(2) = cond_vec1(2) / rlen;

    rlen = std::sqrt(cond_vec2(0)*cond_vec2(0)
                    +cond_vec2(1)*cond_vec2(1)
                    +cond_vec2(2)*cond_vec2(2));

    cond_vec2(0) = cond_vec2(0) / rlen;
    cond_vec2(1) = cond_vec2(1) / rlen;
    cond_vec2(2) = cond_vec2(2) / rlen;

//    check orthonormality
    Real rdot =  cond_vec1(0)*cond_vec2(0) + cond_vec1(1)*cond_vec2(1)
              + cond_vec1(2)*cond_vec2(2);

    if( std::abs(rdot) > 1.0e-5 ) {
        arialog << "[ERROR] Principal directions specified "
                << " are not orthonormal" << dendl;
        return;
    }

    for( Int e = 0; e<nelem; ++e )
    {
      for( Int n=0; n<nint; ++n )
```

```
      {
        elemCond(0, 0, e, n) = cond_data(0);
        elemCond(1, 1, e, n) = cond_data(1);
        elemCond(2, 2, e, n) = cond_data(2);
      // these assignments are not necessary
      //   since the elemCond data is zeroed when subroutine is called
        elemCond(0, 1, e, n) = cond_data(3);
        elemCond(1, 0, e, n) = cond_data(3);
        elemCond(0, 2, e, n) = cond_data(4);
        elemCond(2, 0, e, n) = cond_data(4);
        elemCond(1, 2, e, n) = cond_data(5);
        elemCond(2, 1, e, n) = cond_data(5);

        elemVec(0, 0, e, n) = cond_vec1(0);
        elemVec(0, 1, e, n) = cond_vec1(1);
        elemVec(0, 2, e, n) = cond_vec1(2);

        elemVec(1, 0, e, n) = cond_vec2(0);
        elemVec(1, 1, e, n) = cond_vec2(1);
        elemVec(1, 2, e, n) = cond_vec2(2);
       }
    }

    return;
}

RegisterFunction(CondUserSubC, aucal_aniso_cond, "aucal_aniso_cond");
```

33.22.8 Chemistry Reaction Rate Signature

The call signature for the chemistry reaction rate subroutine is

```
/**
 * Prototype for the chemical reaction rate signature.
 *
 * user_query: query function
 * elem_id: element id
 * reaction_rates_array: reaction rates  (output)
 * ak_array: kinetics coefficients (output)
 * temperature_array: temperature at the gauss points
 * species_array: species array
 * nint: number of gauss points on this element
 * num_species: number of species on this element
 * num_reactions: number of reactions on this element
 * steric_array: steric factors for each reaction
 * prex_array: log pre-exponential factors
 * aenergy_array: activation energies
 * amusp_array: concentration exponents
 */
Int
AcalChemRRUserSubC(
  UserQuery & user_query,
  Int elem_id,
```

```
  ReactionIntegrationPoints reaction_rates_array,
  ReactionIntegrationPoints ak_array,
  ConstIntegrationPoints temperature_array,
  ConstSpeciesIntegrationPoints species_array,
  Int nint,
  Int num_species,
  Int num_reactions,
  ConstReactions steric_array,
  ConstReactions prex_array,
  ConstReactions aenergy_array,
  ConstSpeciesReactions amusp_array);
```

33.22.9 Chemistry Auxiliary Variables Signature

The other chemistry subroutine type is the one that is used to manipulate auxiliary variables. The call signature for this subroutine is as follows

```
/**
 * This is the prototype for the chemical aux variable signature.
 *
 * user_query: query function
 * elem_id: element id
 * temperature_array: temperature at the gauss points
 * species_array:  species array
 * aux_variables_array: auxiliary variable array (output)
 * nint: number of gauss points on this element
 * num_species: number of species on this element
 * num_reactions: number of reactions on this element
 * num_aux_variables:  number of auxiliary variables on this element
 */
Int
AcalChemAVUserSubC(
  UserQuery & user_query,
  Int elem_id,
  ConstIntegrationPoints temperature_array,
  ConstSpeciesIntegrationPoints species_array,
  AuxVariablesIntegrationPoints aux_variables_array,
  Int nint,
  Int num_species,
  Int num_reactions,
  Int num_aux_variables);
```

We note that each of the chemistry subroutines operate on a single element at a time.

33.23 FORTRAN-like C Interface Functions

A number of utility functions are supplied for use in the FORTRAN-like C functions. These utilities provide access to internal variables that one might wish to use in the C functions and enables manipulation of these Real and Integer variables to a certain extent. Examples of interface functions used to modify values would be for assignment of boundary condition, material data or global variables. Additionally, the interface

functions provide access to values associated with data blocks. Details for usage of data block variables with user functions is briefly detailed in the paragraphs which follow.

Within the Aria code data blocks are generically known as *Resources* and the usage of specific *Resources* within the code are referred to as *Region* or *Instance*. Thus the appearance of *Resources*, *Region* or *Instance* in either the name or call signature of the interface function implies usage of a data block.

If an interface function is referenced within the input file with a `USE DATA BLOCK data_block_name` then that data block is in some sense bound to that scope and the data block variables become accessible without explicit reference to the data block name. When the `USE DATA BLOCK DATA data_block_name` specification appears in the Region scope a user subroutine function can access the `data_block_name` variables using *Region* type call signatures. Likewise if the if the `USE DATA BLOCK data data_block_name` specification appears within a material property, boundary condition or volume heat command block then user subroutine functions defined within that command block can access the `data_block_name` variables using *Instance* type call signatures.

A more general usage of data block variables within user subroutine functions requires explicit reference to the data block, a *Resource* argument, within the user interface function call signature. This type of interface call proves useful when one wishes to access data from more than one data block or when one wants to ensure which data block will be used.

When more than one data block will be used within a command block an alternative would be to provide the data block as a `data = data_block_name` argument in the feature specification. For example:
`Heat Transfer Coefficient = Calore_User_Sub name=hcoeff type=element data= data_block_name`.
Of course, one could always employ this approach exclusively when dealing with data block variables in lieu of the `use data block data data_block_name` model usage.

Some of the remaining user interface function calls will reference specific program variables such as time or spatial dimension which cannot be altered (i.e. they will return non-mutable, `const` , program data). In most cases the interface functions will either retrieve values for use in the subroutine or populate values that are communicated back to the code. Here any variables not supplied through the subroutine interface must be explicitly declared within the user subroutine. Access of internal variable values is accomplished using the name associated with the values.

User interface function calls are part of the application code that are accessible to the user subroutine only through the `user_query` facility. Note that many of the access functons are specific to the Aria Region, a material or a feature (i.e. boundary conditions or sources). As an example, in order to reference the the `currentTime()` function the subroutine body would contain a call as indicated here.

```
int
your_subroutine_interface( UserQuery & user_query,
                           more subroutine_arguments, . , . , . , . )
{
  int ierror = 0;
    .
  Real time = user_query.currentTime();
    .
  statements referencing time
    .
  return ierror;
}
```

Call signatures to the various user interface functions are provided below.

- Return the current solution time

```
Real currentTime() const;

- Return the current time step size
Real currentTimeStepSize() const;

- Evaluate a user function ( #values, x, y, function name )
void getFunctionValues( Int, Real *, Real *, const String & name);

- Return Region Data Block values
void getUserRegionData(RealArray1d &real_array, const String & real_label,
                       IntArray1d &int_array, const String & int_label);

- Update Region Data Block values
void putUserRegionData(RealArray1d &real_array, const String & real_label,
                       IntArray1d &int_array, const String & int_label);

- Return Region Data Block Real values
void getUserRealRegionData(RealArray1d &real_array, const String & real_label);

- Update Region Data Block Real values
void putUserRealRegionData(RealArray1d &real_array, const String & real_label);

- Return Region Data Block Integer
void getUserIntRegionData(IntArray1d &int_array, const String & int_label);

- Update Region Data Block Integer values
void putUserIntRegionData(IntArray1d &int_array, const String & int_label);

- Return Feature Data Block values
void getUserInstanceData(RealArray1d &real_array, const String & real_label,
                         IntArray1d &int_array, const String & int_label);

- Update Feature Data Block values
void putUserInstanceData(RealArray1d &real_array, const String & real_label,
                         IntArray1d &int_array, const String & int_label);

- Return Feature Data Block Real values
void getUserRealInstanceData(RealArray1d &real_array, const String & real_label);

- Update Feature Data Block Real values
void putUserRealInstanceData(RealArray1d &real_array, const String & real_label);

- Return Feature Data Block Integer values
void getUserIntInstanceData(IntArray1d &int_array, const String & int_label);

- Update Feature Data Block Integer values
void putUserIntInstanceData(IntArray1d &int_array, const String & int_label);

- Return Material Data Block values
void getUserMaterialData(RealArray1d &real_array, const String & real_label,
                         IntArray1d &int_array, const String & int_label);

- Update Material Data Block values
void putUserMaterialData(RealArray1d &real_array, const String & real_label,
```

```
                        IntArray1d &int_array, const String & int_label);

- Return Material Data Block Real values
void getUserRealMaterialData(RealArray1d &real_array, const String & real_label);

- Update Material Data Block Real values
void putUserRealMaterialData(RealArray1d &real_array, const String & real_label);

- Return Material Data Block Integer values
void getUserIntMaterialData(IntArray1d &int_array, const String & int_label);

- Update Material Data Block Integer values
void putUserIntMaterialData(IntArray1d &int_array, const String & int_label);

- Return Material Data Block Real values
void getUserRealMaterialData(Real &real_value, const String & label);

- Update Material Data Block Real values
void putUserRealMaterialData(Real &real_value, const String & label);

- Return Material Data Block Integer values
void getUserIntMaterialData(Int &int_value, const String & label);

- Update Material Data Block Integer values
void putUserIntMaterialData(Int &int_value, const String & label);

- Retrieve a Real face variable by name to an array
void getRealFaceVar(Int id, const String & name, Real * s);

- Retrieve a Real element variable by name to an array
void getRealElemVar(Int id, const String & name, Real * s);

- Retrieve a Real node variable by name to an array
void getRealNodeVar(Int id, const String & name, Real * s);

- Retrieve an Integer face variable by name to an array
void getIntFaceVar(Int id, const String & name, Int * s);

- Retrieve an Integer element variable by name to an array
void getIntElemVar(Int id, const String & name, Int * s);

- Retrieve an Integer node variable by name to an array
void getIntNodeVar(Int id, const String & name, Int * s);

- Update a Real face variable by name from an array
void putRealFaceVar(Int id, const String & name, Real * s);

- Update a Real element variable by name from an array
void putRealElemVar(Int id, const String & name, Real * s);

- Update a Real node variable by name from an array
void putRealNodeVar(Int id, const String & name, Real * s);

- Update an Integer face variable by name from an array
```

```
void putIntFaceVar(Int id, const String & name, Int * s);

- Update an Integer element variable by name from an array
void putIntElemVar(Int id, const String & name, Int * s);

- Update an Integer node variable by name from an array
void putIntNodeVar(Int id, const String & name, Int * s);

- Retrieve the current time step number
Int currentTimeStepIndex();

- Retrieve the spatial dimension
Int getSpatialDimension();

- Retrieve the material name for the block or sideset being processed
void materialName(String & meshpart_name);

- Retrieve the name of the block or sideset being processed
String getPartName(){ return m_block_name; }

- Retrieve the material name of the contact sideset being processed
void getContactMaterialName(Int, String &);

- Retrieve the material name of the contact sideset being processed
void getContactElement(const Int *, Int *, Int *);

- Retrieve the predicted temperature in this solution step
void nodePredT(Int id, Real *);

- Retrieve the temperature in the previous solution step
void nodePrevT(Int id, Real *);

- Retrieve the temperature time derivative at the Gauss points
void TDotElementGauss(Int id, Real *tDot);

- Retrieve the Real auxiliary variable values
void getAuxVar(Int id, Real * a);

- Set the Real auxiliary variable values
void setAuxVar(AuxVariablesIntegrationPoints aux);

- Return the number of auxiliary variable
Int getNumAuxVars();

- Return the species variables
void getSpecies(Int id, Real * s);

- Return the volume of the current volume element
bool getElementVolume(Int id, Real * value);

- Return the mass of the current volume element
bool getElementMass (Int id, Real * value);

- Return the index of a field within a set of auxiliary variables
```

```
bool getFieldIndex(const Real * fieldPtr, Int * index, const String &name);

- Return a single Encore global variable
Real or Int getEncoreGlobalVariable(const String &name)

- Return a vector Encore global variable
void getEncoreGlobalVariable(Real or Int value, const String &name)

- Return a global variable
Real or Int getGlobalVariable(const String &name)

- Return a global variable to the given value storage
void getGlobalVariable(Real or Int value, const String &name)

- Update the locally stored global variable
void locallyUpdateUserGlobalVariable(Real or Int value, const String &name)

void getUserRealData(RealArray1d &real_array, const String & real_label,
                           const String & resource,
                           const String & user_sub_name );

void putUserRealData(RealArray1d &real_array, const String & real_label,
                           const String & resource, const String & origin,
                           const String & user_sub_name );

void getUserRealData(Real &real_value, const String & real_label,
                           const String & resource,
                           const String & user_sub_name );

void putUserRealData(Real &real_value, const String & real_label,
                           const String & resource, const String & origin,
                           const String & user_sub_name );

void getUserIntData(Int &int_value, const String & int_label,
                          const String & resource,
                          const String & user_sub_name );

void putUserIntData(Int &int_value, const String & int_label,
                          const String & resource, const String & origin,
                          const String & user_sub_name );

void getUserIntData(IntArray1d &int_array, const String & int_label,
                          const String & resource,
                          const String & user_sub_name );

void putUserIntData(IntArray1d &int_array, const String & int_label,
                          const String & resource, const String & origin,
                          const String & user_sub_name );

- Return the local face, element or node currently being processed
void push_query(UserQuery *query)
void pop_query(UserQuery *query)
```

33.23.1 CHEMEQ Species Variable Access

Occasionally one may wish to access the values of species from the CHEMEQ system within the FORTRAN-like C subroutine using the `getSpecies(Int elem_id, Real species *)` interface. In this case some knowledge of the returned data layout is required in order to access the correct values. Internal to the code species values for an element are defined at each of the element quadrature points and stored collectively in a single flat array. Thus in order to access species values for a specific component one must stride into the flat array of values.

As an example, consider the case of three species, num_species = 3, chemistry on a 2D quadrilateral which has four quadrature points. Within the subroutine storage must be provided for an array of length 12 denoted here as s,

RealArray1D s(12);

This implies the species data layout shown below

$$\underbrace{s(0)\ s(1)\ s(2)}_{qpt_1}\ \underbrace{s(3)\ s(4)\ s(5)}_{qpt_2}\ \underbrace{s(6)\ s(7)\ s(8)}_{qpt_3}\ \underbrace{s(9)\ s(10)\ s(11)}_{qpt_4} \quad .$$

We note that if one is interested in the values for the second species these values, $s(1), s(4), s(7), s(10)$, are offset by 1 into each subset of quadrature point values. For a given element id, `eid`, the species values for that element are obtained with the user_query `getSpecies( eid, &s )`. Then to access each of the ith quadrature point values (i=0,1,2,3) for the desired species one must stride into an appropriate offset position, index = 1, so that the species quadrature point value is

value = s(num_species * i + index) .

33.24 Debug Output From User Subroutines

Oftentimes one finds it instructive to output diagnostic information to file from the user subroutine. Aria provides three different ways of generating this output, output directly to a user specified file, output to the log file for all time and selective output to the log file. Each of these options has both advantages and disadvantages. In each of the cases information could be written each time the subroutine is called so the user must provide some code logic to limit the amount of output.

Outputting directly to a user specified file is most straightforward approach but then program output will be split over multiple files. Output to the log file is convenient but in this case the subroutine output will be interlaced with log file output that one would normally expect. Outputting to the log file for all time is appropriate only when diagnostics are always needed. Selective output to the log file is preferred in most cases since the amount of output is restricted to special instances. As an example of each approach the subroutine appearing in a previous section is augmented with the appropriate code. Here direct output to file is demonstrated with both C file specification and C++ file specification.

```
Int
aucal_aniso_cond(
  UserQuery &                          user_query,
  ElementIds                           objid,
  Int                                  nelem,
  Int                                  nint,
  Int                                  spatial_dimension,
  CoordinateElementIntegrationPoints   coords,
  TemperatureElementIntegrationPoints  temperature,
  ConductivityElementIntegrationPoints elemCond,
```

```
  DirectionElementIntegrationPoints     elemVec)
{
   // main ingredients for separate file output

   // C file specification

    static FILE *file_pointer = NULL;
    if(NULL == file_pointer ) // only open the file once
    {
      file_pointer = fopen("c.out", "w");
      ThrowRequire(file_pointer);
    }

    // C++ file specification
    static bool have_ofs = false;
    static std::ofstream ofs;

    if( !have_ofs )
    {
      ofs.open("cpp.out", std::ios::out );
      have_ofs = true;
    }

    RealArray1d cond_data(6);
    RealArray1d cond_vec2(3);
    RealArray1d cond_vec1(3);

    user_query.getUserRealInstanceData(cond_data, "cond_data");
    user_query.getUserRealInstanceData(cond_vec1, "cond_vec1");
    user_query.getUserRealInstanceData(cond_vec2, "cond_vec2");

//     normalize the vectors
    Real rlen = std::sqrt(cond_vec1(0)*cond_vec1(0)
                         +cond_vec1(1)*cond_vec1(1)
                         +cond_vec1(2)*cond_vec1(2));

    cond_vec1(0) = cond_vec1(0) / rlen;
    cond_vec1(1) = cond_vec1(1) / rlen;
    cond_vec1(2) = cond_vec1(2) / rlen;

    rlen = std::sqrt(cond_vec2(0)*cond_vec2(0)
                    +cond_vec2(1)*cond_vec2(1)
                    +cond_vec2(2)*cond_vec2(2));

    cond_vec2(0) = cond_vec2(0) / rlen;
    cond_vec2(1) = cond_vec2(1) / rlen;
    cond_vec2(2) = cond_vec2(2) / rlen;

//    check orthonormality
    Real rdot =  cond_vec1(0)*cond_vec2(0) + cond_vec1(1)*cond_vec2(1)
              + cond_vec1(2)*cond_vec2(2);

// orthonormality diagnosics
    if( std::abs(rdot) > 1.0e-5 ) {
```

```
        // Information is always output to the c.out file C style
        fprintf(file_pointer, "[ERROR] Principal directions specified are not orthonormal.\n");
        // Information is always output to the cpp.out file C++ style
        ofs << "[ERROR] Principal directions specified "
            << " are not orthonormal" << std::endl;
        // Information is always output to log file
        arialog << "[ERROR] Principal directions specified "
                << " are not orthonormal" << dendl;
        // Information is conditionally output to log file by supplying the
        //   -O "-arialog user-sub"   arguments to sierra command
        debuglog << "[ERROR] Principal directions specified "
                 << " are not orthonormal" << dendl;
        return;
    }

    for( Int e = 0; e<nelem; ++e )
    {
      for( Int n=0; n<nint; ++n )
      {
        elemCond(0, 0, e, n) = cond_data(0);
        elemCond(1, 1, e, n) = cond_data(1);
        elemCond(2, 2, e, n) = cond_data(2);
      // these assignments are not necessary
      //   since the elemCond data is zeroed when subroutine is called
        elemCond(0, 1, e, n) = cond_data(3);
        elemCond(1, 0, e, n) = cond_data(3);
        elemCond(0, 2, e, n) = cond_data(4);
        elemCond(2, 0, e, n) = cond_data(4);
        elemCond(1, 2, e, n) = cond_data(5);
        elemCond(2, 1, e, n) = cond_data(5);

        elemVec(0, 0, e, n) = cond_vec1(0);
        elemVec(0, 1, e, n) = cond_vec1(1);
        elemVec(0, 2, e, n) = cond_vec1(2);

        elemVec(1, 0, e, n) = cond_vec2(0);
        elemVec(1, 1, e, n) = cond_vec2(1);
        elemVec(1, 2, e, n) = cond_vec2(2);
       }
    }

    return;
}

RegisterFunction(ElemTensorUserSubC, aucal_aniso_cond, "aucal_aniso_cond");
```

In some instances one might wish to terminate execution when certain conditions arise in the user subroutine. Here the user can provide a diagnostic for termination when causing the program to stop including a conditional such as the one shown below.

```
  if( condvec(2) == 0 )
    throw sierra::RuntimeError() << "condvec(2) = 0 on " << objid;
```

33.25 FORTRAN User Subroutines

An Aria FORTRAN user subroutine must be a text file of suffix .F or .f . The FORTRAN file content is made available to the application by either compiling the module and statically linking it to the application archive or by dynamically linking the module to the executable at run time. The static link approach is often preferred on capability high-performance computing platforms, where dynamically linked subroutines are either not supported or suffer a penalty in performance. User selection of static or dynamic linking is determined by a combination of commands used for job execution and input command lines.

For the static link approach consider the use of a FORTRAN subroutine module file named `Aucal_ConvFlux.F`. In order to use this module the input file must contain an input command line such as

```
user subroutine file = Aucal_ConvFlux.F
```

A static link of the module is then initiated by use the job command

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

This command will invoke creation of an auxiliary FORTRAN module used to perform a registration of the subroutine in the executable. Then the subroutine and auxiliary module are compiled followed by a static link with the existing archive to create an executable written to a `UserSubsProject/bin` subdirectory. This executable must then be specifically called out in the job execution command.

For the dynamic link approach the user must supply the auxiliary FORTRAN module used for subroutine registration. Most often the auxiliary module is included in the subroutine file but can also be provided as a standalone file. In order to use this module the input file must contain an input command line such as

```
load user plugin file Aucal_ConvFlux.so using function conv_flux_register
```

where `conv_flux_register` is the name of the auxiliary module used for subroutine registration. An example of the auxiliary module is given in the section which follows 33.26. Note that this registration is equivalent to that of Aria User Plugin function registration 18. Invocation of the job command

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

initiates a compilation that leads to creation of a shared object, `Aucal_ConvFlux.so`. During execution of the job the shared object will be automatically loaded and used in the execution.

Note that, since `Aucal_ConvFlux.F` is a filename, its input file name is case-sensitive, and considerations must be made concerning its location path. The user is free to name the file and subroutines in any way chosen, provided it is consistent with compiler and operating system limitations. For example, special characters such as '*' and ',' should be avoided. Finally actual use of the function by the application is triggered by association of the the user subroutine name in a particular command line, presently only Dirichlet boundary conditions, flux boundary conditions and material property evaluations.

33.26 FORTRAN Interfaces

While FORTRAN user subroutines could be defined to supply many different parameters, only two different signatures are currently provided in Aria. These signatures characterize the argument lists by which the subroutine interacts with the code. The various subroutine signatures are denoted as:

- node signature 33.26.1
- element signature 33.26.2

Generally speaking FORTRAN subroutines function in the same way as C-Style subroutines with emphasis placed on providing one data set or material property through the element signature interface. Element subroutines operate strictly upon quadrature point values whereas the node subroutine can only operate on nodal values. The different subroutine call signatures are outlined below. The FORTRAN user subroutines must be registered with a function signature similar to the element subroutine example below.

```
c
c  fortran subroutine registration
c
      subroutine conv_flux_register
c
      implicit none
      external aucal_fort_elem_sub
      external ht_coeff_test
      external t_ref_test
c
      call fmwkusersub(aucal_fort_elem_sub, ht_coeff_test,
     &                 'ht_coeff_test')
      call fmwkusersub(aucal_fort_elem_sub, t_ref_test,
     &                 't_ref_test')
c
      end
```

Here the first argument to *fmwkusersub* is the internally defined interface signature name. The second argument is the name of the FORTRAN function (subroutine) and the third is the name to be associated with the subroutine from within the input file. Note that the Fortran function must be declared *external* in order to be recognized by the registration function.

Perhaps the biggest difference between C-Style subroutines and FORTRAN subroutines will be the difference in how problem specific input file data is provided to the user subroutine. For FORTRAN and FORTRAN-like C-style subroutines parameters can be provided through use of arbitrary DATA_BLOCKs 33.19. For boundary conditions, source terms and initial condition the FORTRAN subroutine parameters can also be provided with REAL DATA or INT DATA command lines within the respective command block. However, parameters for material property subroutines can only be provided through the DATA_BLOCK construct. It is important to note that DATA_BLOCK parameter retrieval from within a FORTRAN subroutine is considered to be somewhat awkward and is demonstrated in an example of a material property subroutine.

33.26.1 FORTRAN Node Subroutine

The FORTRAN Node subroutine is used primarily for applying Dirichlet boundary conditions and is designed to operate on sidesets. The Node subroutine is registered in the same way as an element subroutine but with an *aucal_fort_node_sub* signature. For example the subroutine call could be invoked using the following input command block:

```
Begin Temperature Boundary Condition bcblock1
   Temperature fortran subroutine = dirich_test
   Add surface surface_2
End Temperature Boundary Condition bcblock1
```

Here we demonstrate the node user subroutine in which the user supplied subroutine nodal values are assigned as constants. Note that time and coordinates are being supplied to the subroutine so the assigned values could also be functions of the supplied variables.

```
c
c     Subroutine to handle coordinate and time-dependent
c       Temperature Dirichlet BC
c
      subroutine dirich_test(objid,nnodes,coords,temperature,
     &                       time,rdat,idat,ierror)
      implicit none
      integer nnodes
      integer objid
      double precision coords(2,nnodes)
      double precision temperature(nnodes)
      double precision time
      integer ierror

c   Note: rdat and idat must be sized at least 1,
c           even when not used
c         otherwise they must be sized consistent with the
c           data being provided from the input file

      double precision rdat(1)
      integer idat(1)
c
c     INPUT ARGUMENTS:
c        objid(nelem): array of mesh object ID's.  Useful for passing through
c                  to query methods
c        nelem: number of elements in the workset
c        nint:  Number of integration points per element in the workset
c        coords(2,n,nint): array of the coordinates (x,y,z) of the Gauss points
c        rdat() real parameter data
c        idat() integer data
c
c     OUTPUT ARGUMENTS:
c        temperature: temperature at the node
c        ierror: user defined error code for calore to test.
c                zero means success.
c
      integer i, j

      ierror = 0

      do i=1,nnodes

        if( coords(2,i) .lt. 0.5 ) then
           temperature(i) = 1.5
        else
           temperature(i) = 0.1
        end if

      end do

      return
      end
```

33.26.2 FORTRAN Element Subroutine

The FORTRAN element subroutine can be used to compute coefficients or material properties associated with either surface or volume elements. While the example given here it is associated with surface flux conditions, it is important to note that the element subroutine can be more broadly applied.

The FORTRAN element subroutine is used to applying a flux boundary condition and is here defined to operate on sidesets. For example the subroutine call could be invoked using the following input command block:

```
Begin Heat Flux Boundary Condition C1
  Add Surface surface_1
  Flux Fortran Subroutine = ht_flux_test
  integer data 8, 12, 14
  real data 1.0 2.0
end
```

Here we demonstrate the flux user subroutine in which the user supplied subroutine heat flux values are assigned as constants. Note that temperature, time and coordinates are being supplied to the subroutine so the assigned values could also be functions of the supplied variables.

```
c
c     Subroutine to handle coordinate, time and temperature
c        dependent flux
c
      subroutine flux_test(objid,nelem,nint,coords,temperature,
     &                        flux,time,rdat,idat,ierror)
      implicit none
      integer nelem, nint
      integer objid(nelem)
      double precision coords(3,nelem,nint)
      double precision temperature(nelem,nint)
      double precision flux(nelem,nint)
      double precision time
      integer ierror
c
c   Note: rdat and idat must be sized at least 1,
c           even when not used
c         otherwise they must be sized consistent with the
c           data being provided from the input file
c
c         Here the rdat and idat are provided but are not used.
      double precision rdat(2)
      integer idat(3)
c
c    INPUT ARGUMENTS:
c        objid(nelem): array of mesh object ID's.  Useful for passing through
c                  to query methods
c        nelem: number of elements in the workset
c        nint:  Number of integration points per element in the workset
c        coords(3,n,nint): array of the coordinates (x,y,z) of the Gauss points
c        temperature(n,nint): temperature at the Gauss points
c        rdat() real parameter data
```

```
c        idat() integer data
c
c    OUTPUT ARGUMENTS:
c        flux:  array of length ( n, nint)  containing
c                the heat flux coefficient
c                at integration points
c        ierror: user defined error code for calore to test.
c                zero means success.
c
      integer i, j

      ierror = 0

      do i=1,nelem

         do j=1,nint
            if( coords(3,i,j) .gt. 4.0 ) then
               flux(i,j) = 4000.0
            else
               flux(i,j) = 0.0
            end if

         end do

      end do

      return
      end
```

Material model support from within the Aria Material command block for FORTRAN subroutines is currently limited to a few thermal properties, density, specific heat, scalar thermal conductivity and emissivity where the material model designation is simply FORTRAN. Note that the ability to supply FORTRAN subroutine parameters is limited to the use of a DATA BLOCK 33.19.

An example of the material input for a FORTRAN density model subroutine input is shown below followed by the subroutine implemention. In this example the density will take on different constant values above and below a threshold temperature.

```
begin data block my_data
   real real_data = ( 287 1.1 0.9 )
end data block my_data

Begin Aria Material testuser
  Density = Fortran SUB_NAME = my_rho data = my_data
  Thermal Conductivity = constant k = 20.0
  Specific Heat   = CONSTANT cp = 1.0 # J/kg-K
  Heat Conduction = generalized
End Aria Material testuser
```

From the subroutine implementation it becomes clear that DATA_BLOCK parameter retrieval from within a FORTRAN subroutine is somewhat obscure in that the input file named data and values must be explicitly defined within the subroutine to match the input file description. Moreover, the variables *rdat* and *idat* provided through the interface should not be used within the subroutine as storage allocation for those

variables is not provided within the code. For this reason FORTRAN-like C material model subroutines are generally preferred over the FORTRAN counterpart.

```
      Subroutine my_rho(objid,nelem,nint,coords,temperature,
     &                  rho,time,rdat,idat,ierror)
c
c  rdaand idat should not used within a material subroutine
c     s doing so may corrupt internal data
c
      implicit none
      integer nelem,nint
      integer objid(nelem)
      double precision coords(3,nelem,nint)
      double precision temperature(nelem,nint)
      double precision rho(nelem,nint)
      double precision time
      double precision rdat
      double precision idat
      integer ierror

      integer i, j

      double precision temp_limit
      double precision rho_low
      double precision rho_high

      character*9 real_data_name
c                                  3 data values
      double precision data_rdat(3)
c                                  9 characters long
      real_data_name = 'real_data'
c                                  3 data values
c                                  9 characters long
      call aria_get_instance_real_data(data_rdat,3,
     &                                 real_data_name,9)

      temp_limit = data_rdat(1)
      rho_low    = data_rdat(2)
      rho_high   = data_rdat(3)

      ierror = 0

      do i=1,nelem
       do j=1,nint
         if ( temperature(i,j) .lt. temp_limit ) then
          rho(i,j) = rho_low
         else
          rho(i,j) = rho_high
         end if
       end do
      end do

      return
      end
```

33.27 FORTRAN Subroutine Interface Functions

A number of utility functions are supplied for use in the FORTRAN subroutines. These utilities provide access to internal variables that one might wish to use within the subroutine and enables manipulation of these Double Precision and Integer variables to a certain extent. Examples of interface functions used to modify values associated with boundary condition, material data or global variables.

User interface function calls are part of the application code that are accessible from the user subroutine. As an example, in order to retrieve a global variable named *T_AVERAGE* the subroutine body would contain a an external definition of the retrieval function and the function call as indicated here.

```
subroutine your_subroutine_interface( objid,nelem,nint,coords,temperature,
     &                                source,time,rdat,idat,ierror)
      implicit none
      integer nelem, nint
      integer objid(nelem)
      double precision coords(3,nelem,nint)
      double precision temperature(nelem,nint)
      double precision source(nelem,nint)
      double precision time
      integer ierror

      double precision t_avg
      character*9 cvar

      external aria_get_global_real_var

      cvar = "T_AVERAGE"
      ierror = 0
        .
      call aria_get_global_real_var(t_avg,1,cvar)
        .
C     statements referencing t_avg
        .
      return
      end
}
```

Call signatures to the various FORTRAN user interface functions are provided below.

```
      aria_get_time(double precision time)

      aria_get_timestep_index(integer k)

      aria_get_material_name(integer length, character name)

      aria_get_contact_material_name(integer face_id, integer length, character name)

      aria_get_contact_element(integer face_id, integer elem_id, integerfound)

      aria_get_pred_nodal_t(integer id, double precision t, integer found)

      aria_get_prev_nodal_t(integer id, double precision t, integer found)
```

```
aria_get_species(integer id, double precision s)

aria_get_aux_var(integer id, double precision a)

aria_get_element_tdot(integer id, double precision tdot)

aria_get_encore_global_real_var(double precision value,  integer varNameLen,  character varName)

aria_get_encore_global_int_var(integer value,  integer varNameLen,  character varName)

aria_get_global_real_var(double precision value,  integer varNameLen,  character varName)

aria_get_global_int_var(integer value,  integer varNameLen,  character varName)

aria_lupdate_global_real_var( double precision deltaValue,  integer varNameLen,  character varNam

aria_lupdate_global_int_var( integer deltaValue,  integer varNameLen,  character varName)

aria_get_element_volume(integer id, double precision value)

aria_get_element_mass(integer id, double precision value)

aria_get_real_nodal_var(double precision value, integer id, integer varNameLen, character varName

aria_get_int_nodal_var(integer value, integer id, integer varNameLen, character varName)

aria_put_real_nodal_var(double precision value, integer id, integer varNameLen, character varName

aria_put_int_nodal_var(integer value, integer id, integer varNameLen, character varName)

aria_get_real_elem_var(double precision value, integer id, integer varNameLen, character varName)

aria_get_int_elem_var(integer value, integer id, integer varNameLen, character varName)

aria_put_real_elem_var(double precision value, integer id, integer varNameLen, character varName)

aria_put_int_elem_var(integer value, integer id, integer varNameLen, character varName)

aria_get_real_face_var(double precision value, integer id, integer varNameLen, character varName)

aria_get_int_face_var(integer value, integer id, integer varNameLen, character varName)

aria_put_real_face_var(double precision value, integer id, integer varNameLen, character varName)

aria_put_int_face_var(integer value, integer id, integer varNameLen, character varName)

aria_get_material_real_data(double precision addr,  integer length,  character name,
                            integer name_length)

aria_put_material_real_data(double precision addr,  integer length,  character name,
                            integer name_length)

aria_get_material_int_data(integer addr,  integer length,  character name,
```

```
                              integer name_length)

aria_put_material_int_data(integer addr,  integer length,  character name,
                              integer name_length)

aria_get_region_real_data(double precision addr,  integer length,  character name,
                             integer name_length)

aria_put_region_real_data(double precision addr,  integer length,  character name,
                             integer name_length)

aria_get_region_int_data(integer addr,  integer length,  character name,
                            integer name_length)

aria_put_region_int_data(integer addr,  integer length,  character name,
                            integer name_length)

aria_get_instance_real_data(double precision addr,  integer length,  character name,
                              integer name_length)

aria_put_instance_real_data(double precision addr,  integer length,  character name,
                              integer name_length)

aria_get_instance_int_data(integer addr,  integer length,  character name,
                              integer name_length)

aria_put_instance_int_data(integer addr,  integer length,  character name,
                              integer name_length)

aria_get_spatial_dimension(integer spatDim)

aria_evaluate_function( integer numValues, double precision input, double precision output,
                         character name,  integer name_length)
```

33.28 Solution Mapping Methodology

Numerical simulations of full system models are oftentimes intensely consuming calculations. During the design process the need often arises to study the same problem with minor changes to a subcomponent of the original model. To fulfill this need a solution mapping methodology that simulates the thermal response of a subassembly, without having to repeat a simulation of the full assembly model has been developed. This capability is particularly useful in cases where the analyst must evaluate alternative subassembly designs and provide timely input to engineering design studies. This section describes how to apply this solution mapping methodology.

The basic idea of the Aria solution mapping methodology is to use existing simulation results (e.g temperature or heat flux) from a full assembly model to set boundary conditions (BCs) for a separate simulation of a smaller, but possibly more geometrically resolved, subassembly or component. In effect, the Aria thermal simulation of the full assembly serves as the "Donor" of the solution results, while the subassembly model becomes the "Recipient". This methodology uses existing Aria code features to map solution results to the appropriate boundary surfaces of the subassembly model. It is important to mention that the methodology permits the analyst to use different meshes and mesh resolutions at the interface between the full and

subassembly models.

33.28.1 Setting up Simulation for Donor Model

To set up a Donor model, the user modifies the Aria input file for the full assembly model so that it outputs the solution information that will be transferred or mapped to the outer surfaces of the Recipient subassembly model or mesh. The user can choose to set either Dirichlet boundary conditions (i.e. map temperature) or Neumann type boundary conditions (i.e. map heat fluxes) on the outer surface of the recipient model. Numerical studies conducted by the Aria development team indicate that temperature mapping generally provides much more reliable estimation of the subassembly temperatures than using heat flux mapping. In fact, the user is cautioned to avoid the flux mapping transfer capability, especially for long-duration transients when the error accumulates in time and can become quite substantial.

In either type of solution mapping, the user must compile and specify the material blocks surrounding and contacting the subassembly, which are to be used in the solution mapping. It is recommended that, where possible, the user should map the solution from the Donor model to an enclosing region that extends slightly beyond the subassembly, rather than mapping solutions directly on the subassembly exterior. This approach provides some measure of buffering of the effect of sudden changes in time varying BCs.

First, add a new "Begin Results Output" (described in 25.1) block to the input file used specifically for mapping, such as the example shown below. This output block is in addition to the typical output block that writes the nodal field for the entire model domain.

```
  .
  .
  .
       Begin Results Output Solution_Mapping
          Database Name = Donor_solution.e
          Include = BlockList
#         ... Use for Temperature Mapping Only ...
          Nodal Variables = solution->Temperature as T
#         ... Use for Heat Flux Mapping Only ...
          Nodal variables = pp->Heat_Conduction as  Heat_Flux
          At Time 0.0, Increment = 1.0
          Timestep Adjustment Interval is 4
       End
  .
  .
```

In the above syntax example, the *BlockList* is the list of block IDs that will be used in the solution mapping. If heat flux mapping is used, then add the following command to the "Region" level of the input file:

```
  .
       Postprocess Heat_Conduction on all_blocks
  .
```

With above command lines added, execute the thermal simulation for the full assembly model and create the donor-solution exodus output file.

33.28.2 Setting up Simulation for Recipient Model

In most cases, the user can prepare the Aria input file for the Recipient model by copying and simplifying the input file from the original full assembly model. The input file for Recipient model will reference the subassembly mesh and the subassembly material definitions. With the input file stylized for the subassembly, the user should add:

1. A "Begin Finite Element Model" block
2. A "Begin Input_Output Region" block
3. Transfer Commands
4. Transfer definitions via the "Begin Transfer" block
5. BC XFER command (for temperature mapping only).

For the case where heat flux is used in the solution mapping, the last step is replaced with:

1. Addition of the user field command (see Chapter 9 and
2. Addition of a "Flux Vector Node Variable" (see Chapter 33.1)

1. Finite Element Model

The purpose of the new Finite Element Block is to identify the exodus file generated by the Donor model. The syntax of this command block is described in Chapter 2. An example of this block is shown below:

```
Begin Finite Element Model Input_Transfer
     Database Name = Donor_solution.e
     Database Type = exodusII
End
```

This block is placed in the Domain level of the input file. Explanation of Domain, Procedure and Region levels of the input file can be found in Chapter 2 of the manual.

2. Begin Input_Output Block

A description of the "Begin Input_Output Region" block can be found in Section 15.1. This command block is placed in the Procedure level of the input file. An example is given below:

```
Begin Input_Output Region Transfer_Region
     Use Finite Element Model Input_Transfer
End
```

3. Transfer Commands

A detailed explanation of Transfer command syntax and usage is presented in Section 15.1 of this user manual. These commands are added to the "Begin System Main" block. An example of these transfer commands is shown below:

```
Begin Solution Control Description
    Use System Main
    Begin System Main
        Simulation Start Time = 0.0
        Simulation Termination Time = 3600.0
        Begin Transient solution_block_1
            Advance transfer_region
            Transfer  trans_to_aria
            Advance  myRegion
        End
       .
       .
   End System Main
        .
        .
End Solution Control Description
```

The above command block is located in the Procedure level of the input file.

4. Transfer Definitions

Syntax and usage of transfer definitions is presented in Section 15.1 of the manual. Here is an example of the transfer definitions for

```
      Begin Transfer trans_to_aria
           Interpolate Volume Nodes from transfer_region to myRegion
           Nodes Outside Region = project   #or extrapolate, #or truncate
#          ... Use for Temperature transfer only ...
           Send Block block_1 block_2 ...  to  surface_1 surface_2 ...
           Send Field T State None to solution->Temperature State New
#          ... Use for heat flux transfer only ...
#          Send Block block_1 block_2 ...  to  block_1 block_2 ...
#          Send Field heat_flux State None to surf_flux State None
#          ... Tolerance settings ...
           Search Geometric Tolerance is 0.001
           Search Surface Gap Tolerance is 0.01
  End
```

The above example has transfer definitions for both temperature and heat flux, however, only one set of commands can be used.

5a. BC XFER Command

As explained previously, the BC XFER command is used to set the temperature BC on the Recipient model. The command is placed in the Region level of the input file. Here is an example of the command:

```
    BC XFER Dirichlet on surface_1 ... Temperature    #BC sidesets for recipient mesh
```

General usage and syntax for the BC XFER command can be found in Chapter 9 of the manual.

5b. Heat Flux BC Commands

For the case where heat flux mapping is used, the above BC XFER command is replaced by: (1) user field definition and (2) heat flux BC specification. Syntax for the user field definition is presented in Chapter 9 of the manual, while its use in a heat flux BC specification is given in Chapter 33.1. Here are examples of the applicable commands:

```
#       ... Create User Field to store mapped heat flux ...
        User Field Real Nodal Vector surf_flux on block_1
```

The above command must be added to the Region level of the input file. In turn, the *surf_flux* is used to set the heat flux BC via the command block:

```
#    ... Set the heat flux BC ...
     Begin Heat Flux Boundary Condition mapped_flux
       add Surface surface_1
       Flux Vector Node Variable = surf_flux
     End
```

Chapter 34

Correlation Heat Transfer Coefficient Reference

34.1 Heat Transfer Coefficient Correlation Library

For heat transfer problems involving convective flux boundary conditions, a convection heat transfer coefficient must be specified (Ch 33.1). An extensive built-in heat transfer coefficient correlation library has been incorporated into Aria (see [42]) for generalized heat transfer problems involving fluid flow. The current code implementation includes only subset of this library but future plans are to provide full support of those correlations. This library includes correlations for laminar and turbulent flows of forced and free convection, for internal and external flow geometries, and for gases and liquids (including liquid metals). These correlations are especially useful for reduced order flow modeling in which a full Conjugate Heat Transfer simulation is not desired.

Each correlation supplies a Nusselt Number (Nu) from which the heat transfer coefficient (h) may be computed via

$$h = \frac{K}{D_h} * Nu \tag{34.1}$$

where K is the scalar thermal conductivity and D_h is a characteristic length for the simulation. Hence in all cases, K and D_h MUST be specified. The parameter K is supplied in the fluid "Aria Material" command block and D_h in the "Heat Transfer Correlation Coefficient" command block. In applying the convective flux, provision is made for switching between laminar and turbulent correlations based on a user specification of the transition Reynolds Number (R_{ex}).

Correlations are implemented with various mathematical forms, required input variables and valid flow regimes. A sample input for the correlation heat transfer coefficient is shown below.

```
Begin Heat Transfer Correlation Coefficient ziggurat
  LAMINAR   CORRELATION = type 18
  TURBULENT CORRELATION = type 18

  Transition Reynolds Number = 400

  Characteristic Length = 0.1
  Entrance       Length = 0.5

  Compute Reynolds Number
  Compute Prandtl  Number
  Compute Wall Temperature
  Compute Fluid Temperature
  Compute Friction Factor model = smooth tube
End
```

Contents of the input command block correspond to entries listed in correlation Classification 34.1.1 and the D_h parameter corresponds to the "characteristic length" entry. Parsed contents of the input command block are verified against the "Dependence" requirements of the selected correlation Type 34.1.2 and any missing requirements are reported in the log file.

34.1.1 Classification

Due to the large number of correlations, a taxonomy based on the required input parameters was developed in order to provide a quick mapping between a model and it's index ID. There are currently 20 parameters which all the correlations are dependent on and these are

f	Friction factor
Re	Flow Reynolds number
Pr	Flow Prandtl number
Pr_w	Flow Prandtl number evaluated with wall conditions
T	Fluid temperature
T_w	Wall temperature
L	Wall length
g	Gravitational constant
r	Ratio of annulus inner diameter to outer diameter
ρ	Fluid density
μ	Dynamic viscosity
ν	Kinematic viscosity
β	Expansion coefficient
θ	Wall angle (in radians)
V	Scalar flow velocity magnitude
Re_x	Transitional Reynolds number
S_t	Transverse pitch
S_l	Longitudinal pitch
ϕ	Porosity
D_e	Entrance length

These may be computed analytically from the model variables or overridden by a user specified constant. It is to be noted that specification of a parameter constant means that other parameters which are dependent on it will use this value e.g constant kinematic viscosity will also be used in Reynolds number calculations instead of a material property evaluation.

Each correlation specification description below is of the form

Specification : {Regime, Phase, Convection, Flow, Author, Shape, Special}

where the component options are

Regime: {Laminar, Turbulent, Both}

Phase: {Gas, Liquid, Gas|Liquid, LiquidMetal, Organic}

Convection: {Forced, Free}

Flow: {Internal, External}

Author: {Hausen, Modified_Sieder-Tate, Churchill-Bernstein, Petukhov, Colburn, Dittus-Boelter, Sieder-Tate, Taylor, Nusselt, Notter-Sliecher, Seban-Shimazaki, Azer-Chao, Sliecher, Denton, Jeschar, Achenbach, Beek, Whitaker, Witte, Ranz-Marshall, Modified_Ranz-Marshall, Hilpert, Zhulkauskas, Churchill-Chu, McAdams, Morgan, Gnielinski, Modified_Gnielinski}

Shape: {NoShape, Cylinder, Sphere, Cube, Square_CornerOn, Square_FlatOn, Hexagon_CornerOn, Hexagon_FlatOn, Flat_Plate_Vertical, TubeBank_Aligned_Cylinder, TubeBank_Staggered_Cylinder, Flat_Plate, Vertical_Or_Angle_Surface, Vertical_Surface, Horizontal_Plate_OnHC, Horizontal_Plate_OffHC, Horizontal_Cylinder, Annulus}

Special: {N/A, Crossflow, Local, Average, PackedBed, FullDevLam_ConstFlux, FullDevLam_ConstTemp, ConstFlux, ConstTemp, Insulated_InnerWall, Insulated_OuterWall}

34.1.2 Correlation Listing

Beta Capability: Correlation Types 25 and 71-74 are the most tested correlations. All other correlations have not been extensively tested and should be used with caution.

Type 1 This correlation applies to cases of fully-developed internal laminar flow with a constant wall flux or constant wall temperature, for which a Nusselt number may be determined analytically.

Dependence: { K, D_h }
Specification: { Laminar, Gas|Liquid, Forced, Internal, NoAuthor, NoShape, FullDevLam_ConstFlux }
Correlation: $Nu = 4.36$
Range: { $Pr > 0.6$ }

Type 2 This correlation applies to cases of fully-developed internal laminar flow with a constant wall flux or constant wall temperature, for which a Nusselt number may be determined analytically.

Dependence: { K, D_h }
Specification: { Laminar, Gas|Liquid, Forced, Internal, NoAuthor, NoShape, FullDevLam_ConstFlux }
Correlation: $Nu = 3.666$
Range: {$Pr > 0.6$}

Type 3 This correlation by Hausen applies to cases of thermally fully-developed laminar flow with entrance effects.

Dependence: { K, D_h, D_e, Re, Pr }
Specification: {Laminar, Gas|Liquid, Forced, Internal, Hausen, NoShape, N/A}
Correlation:

$$\begin{aligned} I &= \frac{D_h}{D_e}\mathrm{RePr} \\ Nu &= 3.66 + \frac{0.0668 \cdot I}{1 + 0.4 \cdot I^{0.66667}} \end{aligned}$$

.
Range: $\{Pr >> 1\}$ or unheated entrance

Type 4 This correlation by Nusselt applies to cases of thermally fully-developed turbulent flow with entrance effects.

Dependence: { K, D_h, D_e, Re, Pr }
Specification: {Turbulent, Gas|Liquid, Forced, Internal, Nusselt, NoShape, N/A}
Correlation: $Nu = 0.036 Re^{0.8} Pr^{0.3333} \left(\frac{D_h}{D_e}\right)^{0.05556}$

Type 5 This modified Ranz-Marshall correlation applies to cases of thermally fully-developed flow with entrance effects.

Dependence: { K, D_h, D_e, Re, Pr }
Specification: {Both, Liquid, Forced, External, Modified_Ranz-Marshall, NoShape, N/A}
Correlation: $Nu = 2 + 0.6\sqrt{Re}(Pr)^{0.3333} \cdot 25(D_e/D_h)^{-0.7}$

Type 14 This correlation by Azer-Chao applies to cases of constant surface temperature liquid metals in tubes.

Dependence: { K, D_h, Re, Pr }
Specification: {Turbulent, LiquidMetal, Forced, Internal, Azer-Chao, NoShape, N/A}
Correlation: $\mathrm{Pe} = \mathrm{RePr}$, $\mathrm{Nu} = 5.0 + 0.05\mathrm{Pe}^{0.8}\mathrm{Pr}^{0.25}$

Type 15 This correlation by Sliecher applies to cases of constant surface temperature liquid metals in tubes.

Dependence: { K, D_h, Re, Pr }
Specification: {Both, LiquidMetal, Forced, Internal, Sliecher, NoShape, N/A}
Correlation: $\mathrm{Pe} = \mathrm{RePr}$, $\mathrm{Nu} = 4.8 + 0.0156\mathrm{Pe}^{0.85}\mathrm{Pr}^{0.08}$
Range: $\{0.004 < Pr < 0.1\}$, $\{Re < 500000\}$

Type 18 This correlation applies to cases of flow over a sphere.

Dependence: { K, D_h, Re, Pr }
Specification: {Both, Gas, Forced, External, NoAuthor, Sphere, N/A}
Correlation: $\mathrm{Nu} = 0.37\mathrm{Re}^{0.6} \cdot 1.126\mathrm{Pr}^{0.3333}$
Range: $\{17 < Re < 700000\}$

Type 21 This correlation by Zhulkauskas applies to cases of aligned cylindrical tube bank in cross flow.

Dependence: { K, D_h, Re, Pr }
Specification: {Both, Gas, Forced, External, Zhulkauskas, TubeBank_Aligned_Cylinder, Crossflow}
Correlation: $\mathrm{Nu} = C_1 \mathrm{Re}^{C_2} \mathrm{Pr}^{0.36}$

$$(C_1, C_2) = \begin{cases} (0.27, 0.63) & \mathrm{Re} < 200000 \\ (0.021, 0.84) & \text{else} \end{cases}$$

Range: $\{1 < Re < 1000000\}$, $\{0.7 < Pr < 500\}$

Type 23 This correlation by Petukhov applies to cases of turbulent gas flow with friction factor correlation for smooth surfaces.

Dependence: { K, D_h, Re, Pr, f }
Specification: {Turbulent, Gas, Forced, Internal, Petukhov, NoShape, N/A}
Correlation: $\mathrm{Nu} = \frac{0.125 f \mathrm{RePr}}{1.07 + 12.7\sqrt{0.125 f}\left(\mathrm{Pr}^{0.6667} - 1\right)}$
Range: $\{3000 < Re < 5000000\}$, $\{0.5 < Pr < 2000\}$

Type 25 This correlation by Dittus-Boelter applies to cases of turbulent flow.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Turbulent, Gas|Liquid, Forced, Internal, Dittus-Boelter, NoShape, N/A}
Correlation: $\mathrm{Nu} = 0.023\mathrm{Re}^{0.8}\mathrm{Pr}^{e}$

$$e = \begin{cases} 0.3 & T_w < T \\ 0.4 & \text{else} \end{cases}$$

Range: $\{10000 < Re < 1000000\}$, $\{0.7 < Pr < 160\}$

Type 26 This correlation by Sieder-Tate applies to cases of turbulent gas flow.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Turbulent, Gas, Forced, Internal, Sieder-Tate, NoShape, N/A}
Correlation: $\mathrm{Nu} = 0.027\mathrm{Re}^{0.8}\mathrm{Pr}^{0.3333}\left(\frac{T}{T_w}\right)^{0.098}$
Range: $\{10000 < Re < 1000000\}$, $\{0.7 < Pr < 160\}$

Type 27 This correlation by Whitaker applies to cases of gas flow over a sphere.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Both, Gas, Forced, External, Whitaker, Sphere, N/A}
Correlation: $\mathrm{Nu} = 2 + (0.4\sqrt{\mathrm{Re}} + 0.06\mathrm{Re}^{0.6667})\mathrm{Pr}^{0.4}\left(\frac{T}{T_w}\right)^{0.175}$
Range: $\{3.5 < Re < 76000\}$, $\{0.71 < Pr < 380\}$

Type 28 This correlation by Hilpert applies to cases of cylinder in cross flow.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Both, Gas, Forced, External, Hilpert, Cylinder, Crossflow}
Correlation: $\mathrm{Re} = \mathrm{Re}\left(\frac{T}{T_f}\right)^{0.7}$, $\mathrm{Nu} = C_1\mathrm{Re}^{C_2}\mathrm{Pr}^{0.3333}$

$$(C_1, C_2) = \begin{cases} (0.989, 0.33) & \mathrm{Re} < 4 \\ (0.911, 0.385) & \mathrm{Re} < 40 \\ (0.683, 0.466) & \mathrm{Re} < 4000 \\ (0.193, 0.618) & \mathrm{Re} < 40000 \\ (0.027, 0.805) & \text{else} \end{cases}$$

Range: $\{0.4 < Re < 400000\}$

Type 34 This correlation by Zhulkauskas applies to cases of cylinder in cross flow.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Both, Gas, Forced, External, Zhulkauskas, Cylinder, Crossflow}
Correlation: $T_f = \frac{T+T_w}{2}$, $\mathrm{Re} = \mathrm{Re}\left(\frac{T}{T_f}\right)^{0.7}$, $\mathrm{Nu} = C_1\mathrm{Re}^{C_2}\mathrm{Pr}^{C_3}$

$$(C_1, C_2) = \begin{cases} (0.989, 0.33) & \mathrm{Re} < 4 \\ (0.911, 0.385) & \mathrm{Re} < 40 \\ (0.683, 0.466) & \mathrm{Re} < 4000 \\ (0.027, 0.805) & \text{else} \end{cases}$$

$$C_3 = \begin{cases} 0.36 & \mathrm{Pr} > 10 \\ 0.37 & \text{else} \end{cases}$$

Range: $\{1 < Re < 1000000\}$, $\{0.7 < Pr < 500\}$

Type 35 This correlation by Churchill-Bernstein applies to cases of cylinder in cross flow.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Both, Gas, Forced, External, Churchill-Bernstein, Cylinder, Crossflow}
Correlation: $T_f = \frac{T+T_w}{2}$, $\mathrm{Re} = \mathrm{Re}\left(\frac{T}{T_f}\right)^{0.7}$, $\mathrm{Nu} = 0.3 + \frac{0.62\sqrt{\mathrm{Re}}\mathrm{Pr}^{0.3333}}{(1+(0.4/\mathrm{Pr})^{0.6667})^{0.25}}(1 + (\mathrm{Re}/282000)^{0.625})^{0.8}$
Range: $\{0.2 < Re * Pr\}$

Type 36 This correlation applies to cases of flow over a flat plate.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Laminar, Gas, Forced, External, NoAuthor, Flat_Plate, Local}
Correlation: $T_f = \frac{T+T_w}{2}$, $Re = Re \cdot (T/T_f)^{0.7}$, $Nu = 0.332\sqrt{Re}Pr^{0.3333}$
Range: $\{0.6 < Pr\}$

Type 37 This correlation applies to cases of flow over a flat plate.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Laminar, Gas, Forced, External, NoAuthor, Flat_Plate, Average}
Correlation: $T_f = \frac{T+T_w}{2}$, $Re = Re \cdot (T/T_f)^{0.7}$, $Nu = 0.664\sqrt{Re}Pr^{0.3333}$
Range: $\{0.6 < Pr\}$

Type 38 This correlation applies to cases of flow over a flat plate.

Dependence: { K, D_h, Re, Pr, T, T_w }
Specification: {Turbulent, Gas, Forced, External, NoAuthor, Flat_Plate, N/A}
Correlation: $T_f = \frac{T+T_w}{2}$, $Re = Re \cdot (T/T_f)^{0.7}$, $Nu = 0.0296Re^{0.8}Pr^{0.3333}$
Range: $\{0.6 < Pr < 60\}$

Type 39 This correlation by Sieder-Tate applies to cases of turbulent liquid flow.

Dependence: { K, D_h, Re, Pr, T, T_w, μ }
Specification: {Turbulent, Liquid, Forced, Internal, Sieder-Tate, NoShape, N/A}
Correlation: $\mu_w = \left(\mu^{-0.2661} + \frac{T_w - T}{37.073}\right)^{-3.758}$, $\mathrm{Nu} = 0.027\mathrm{Re}^{0.8}\mathrm{Pr}^{0.3333}\left(\frac{\mu}{\mu_w}\right)^{0.14}$
Range: $\{10000 < Re < 1000000\}$, $\{0.7 < Pr < 160\}$

Type 58 This correlation by Churchill-Chu applies to free convection from a vertical surface, or non-vertical surface if angle is less than 60° (measured from vertical).

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ, θ }
Specification: {Laminar, Gas, Free, External, Churchill-Chu, Vertical_Or_Angle_Surface, N/A}
Correlation:

$$\begin{aligned} x &= L^3, \beta = \frac{1}{T_f}, T_f = \frac{T + T_w}{2}, \rho_f = \rho\frac{T}{T_f}, \mu_f = \mu(T_f/T)^{0.7} \\ Gr &= gcos(\theta)x(\rho_f)^2\frac{|T_w - T|}{T_f\mu_f^2}, Ra = GrPr \\ Nu &= 0.68 + 0.67\frac{Ra^{0.25}}{(1 + (0.492/Pr)^{0.565})^{0.4444}} \end{aligned}$$

Range: $\{Ra < 1e^9\}$, $\{\theta < 60°\}$

Type 59 This correlation by Churchill-Chu applies to free convection from a vertical surface, or non-vertical surface if angle is less than 60° (measured from vertical).

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ, θ, β }
Specification: {Laminar, Liquid, Free, External, Churchill-Chu, Vertical_Or_Angle_Surface, N/A}
Correlation:

$$\begin{aligned} x &= \beta L^3, T_f = \frac{T+T_w}{2}, \mu_f = \left(\mu^{-0.2661} + \frac{T_f - T}{37.073}\right)^{-3.758}, Pr = Pr \cdot \mu_f/\mu \\ Gr &= gx(\rho)^2 \frac{|T_w - T|}{T_f \mu_f^2}, Ra = GrPr \\ Nu &= 0.68 + 0.67 \frac{Ra^{0.25}}{(1 + (0.492/Pr)^{0.565})^{0.4444}} \end{aligned}$$

Range: $\{Ra < 1e^9\}$, $\{\theta < 60°\}$

Type 60 This correlation by Churchill-Chu applies to free convection from a vertical surface.

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ }
Specification: {Both, Gas, Free, External, Churchill-Chu, Vertical_surface, N/A}
Correlation:

$$\begin{aligned} x &= L^3, \beta = \frac{1}{T_f}, T_f = \frac{T+T_w}{2}, \rho_f = \rho \frac{T}{T_f}, \mu_f = \mu (T_f/T)^{0.7} \\ Gr &= gx(\rho_f)^2 \frac{|T_w - T|}{T_f \mu_f^2}, Ra = GrPr \\ Nu &= \left(0.825 + \frac{0.387 Ra^{0.16667}}{(1 + (0.492/Pr)^{0.5625})^{0.2963}}\right)^2 \end{aligned}$$

Type 61 This correlation by McAdams applies to free convection from a horizontal surface; either the upper surface of a heated plate or the lower surface of a cooled plate.

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ }
Specification: {Both, Gas, Free, External, McAdams, Horizontal_Plate_OnHC, N/A}
Correlation:

$$x = L^3, \beta = \frac{1}{T_f}, T_f = \frac{T+T_w}{2}, \rho_f = \rho \frac{T}{T_f}, \mu_f = \mu (T_f/T)^{0.7}, Gr = gx(\rho_f)^2 \frac{|T_w - T|}{T_f \mu_f^2}, Ra = GrPr$$

$$Nu = \begin{cases} 0.54 Ra^{0.25} & Ra < 1e^7 \\ 0.15 Ra^{0.3333} & \text{else} \end{cases}$$

Range: $\{1e^4 < Ra < 1e^{11}\}$

Type 62 This correlation by McAdams applies to free convection from a horizontal surface; either the upper surface of a cooled plate or the lower surface of a heated plate.

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ }
Specification: {Both, Gas, Free, External, McAdams, Horizontal_Plate_OffHC, N/A}
Correlation:

$$\begin{aligned} x &= L^3, \beta = \frac{1}{T_f}, T_f = \frac{T+T_w}{2}, \rho_f = \rho \frac{T}{T_f}, \mu_f = \mu (T_f/T)^{0.7}, Gr = gx(\rho_f)^2 \frac{|T_w - T|}{T_f \mu_f^2}, Ra = GrPr \\ Nu &= 0.27 Ra^{0.25} \end{aligned}$$

Range: $\{1e^5 < Ra < 1e^{10}\}$

Type 63 This correlation by Morgan applies to free convection from a horizontal cylinder.

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ }
Specification: {Both, Gas, Free, External, Morgan, Horizontal_Cylinder, N/A}
Correlation:

$$\begin{aligned} x &= L^3, \beta = \frac{1}{T_f}, T_f = \frac{T+T_w}{2}, \rho_f = \rho\frac{T}{T_f}, \mu_f = \mu(T_f/T)^{0.7}, Gr = gx(\rho_f)^2\frac{|T_w - T|}{T_f\mu_f^2}, Ra = GrPr \\ Nu &= C_1 Ra^{C_2} \end{aligned}$$

$$(C_1, C_2) = \begin{cases} (0.675, 0.058) & \text{Ra} < 1e^{-2} \\ (1.02, 0.148) & \text{Ra} < 100 \\ (0.85, 0.188) & \text{Ra} < 10000 \\ (0.48, 0.25) & \text{Ra} < 1e^7 \\ (0.125, 0.333) & \text{else} \end{cases}$$

Range: $\{1e^{-10} < Ra < 1e^{12}\}$

Type 64 This correlation by Churchill-Chu applies to free convection from a horizontal cylinder.

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ }
Specification: {Both, Gas, Free, External, Churchill-Chu, Horizontal_Cylinder, N/A}
Correlation:

$$\begin{aligned} x &= L^3, \beta = \frac{1}{T}, T_f = \frac{T+T_w}{2}, \rho_f = \rho\frac{T}{T_f}, \mu_f = \mu(T_f/T)^{0.7}, Gr = gx(\rho_f)^2\frac{|T_w - T|}{T_f\mu_f^2}, Ra = GrPr \\ Nu &= \left(0.6 + \frac{0.387Ra^{0.16667}}{(1+(0.559/Pr)^{0.5625})^{0.2963}}\right)^2 \end{aligned}$$

Range: $\{1e^{-5} < Ra < 1e^{12}\}$

Type 65 This correlation by Churchill-Chu applies to free convection from a sphere.

Dependence: { K, D_h, Pr, T, T_w, L, g, ρ, μ }
Specification: {Both, Gas, Free, External, Churchill-Chu, Sphere, N/A}
Correlation:

$$\begin{aligned} x &= L^3, \beta = \frac{1}{T}, T_f = \frac{T+T_w}{2}, \rho_f = \rho\frac{T}{T_f}, \mu_f = \mu(T_f/T)^{0.7}, Gr = gx(\rho_f)^2\frac{|T_w - T|}{T_f\mu_f^2}, Ra = GrPr \\ Nu &= 2 + \frac{0.589Ra^{0.25}}{(1+(0.469/Pr)^{0.5625})^{0.4444}} \end{aligned}$$

Range: $\{1e^{11} < Ra, 0.7 < Pr\}$

Type 72 This correlation by Gnielinski applies to cases of turbulent gas flow in a circular tube with friction factor correlation for smooth surfaces.

Dependence: { K, D_h, Re, Pr, f }
Specification: {Turbulent, Gas, Forced, Internal, Gnielinski, Horizontal_Cylinder, N/A}
Correlation: $\text{Nu} = \frac{\frac{f}{8}(\text{Re-1000})\text{Pr}}{1+12.7\sqrt{\frac{f}{8}}\left(\text{Pr}^{0.6667}-1\right)}$
Range: $\{3000 < Re < 5000000\}$, $\{0.5 < Pr < 2000\}$

Type 73 This modified correlation by Gnielinski applies to cases of turbulent flow in an annulus with friction factor correlation for annular surfaces where the inner wall is heated and the outer wall is insulated.

Dependence: { K, D_h, Re, Pr, f, r, T, T_w, Pr_w }
Specification: {Turbulent, Gas|Liquid, Forced, Internal, Modified_Gnielinski, Annulus, Insulated_OuterWall}
Correlation:

$$F = 0.75r^{-0.17}, b = 1.07 + \frac{900}{Re} - \frac{0.63}{1+10Pr}$$

$$Nu = cF\frac{\frac{f}{8}\mathrm{RePr}}{b + 12.7\sqrt{\frac{f}{8}}(\mathrm{Pr}^{0.6667} - 1)}$$

$$c = \begin{cases} 1 & \text{GAS and T} > T_w \\ \left(\frac{T}{T_w}\right)^{0.45} & \text{GAS and T} < T_w \\ \left(\frac{Pr}{Pr_w}\right)^{0.11} & \text{LIQUID} \end{cases}$$

Range: $\{10000 < Re\}$

Type 74 This modified correlation by Gnielinski applies to cases of turbulent flow in an annulus with friction factor correlation for annular surfaces where the inner wall is insulated and the outer wall is heated.

Dependence: { K, D_h, Re, Pr, f, r, T, T_w, Pr_w }
Specification: {Turbulent, Gas|Liquid, Forced, Internal, Modified_Gnielinski, Annulus, Insulated_InnerWall}
Correlation:

$$F = 0.9 - 0.15r^{0.6}, b = 1.07 + \frac{900}{Re} - \frac{0.63}{1+10Pr}$$

$$Nu = cF\frac{\frac{f}{8}\mathrm{RePr}}{b + 12.7\sqrt{\frac{f}{8}}(\mathrm{Pr}^{0.6667} - 1)}$$

$$c = \begin{cases} 1 & \text{GAS and T} > T_w \\ \left(\frac{T}{T_w}\right)^{0.45} & \text{GAS and T} < T_w \\ \left(\frac{Pr}{Pr_w}\right)^{0.11} & \text{LIQUID} \end{cases}$$

Range: $\{10000 < Re\}$

34.1.3 Usage

The convection correlations can be used in two distinct use cases, forced convection through a channel and free convection heat transfer. In the case of forced convection the calculation of flow velocity in the channel coupled with the computed bulk/reference temperature of the fluid enables characterization of the flow. This enables the evaluation of Reynolds number and Prandtl number using the average film temperature. These two parameters are then used in the chosen forced flow correlation. An example input for the convection boundary condition is shown below.

```
begin convective flux boundary condition insideWall
  add surface surface_2
  use advective bar airBar bulknodes
  USE CORRELATION CONVECTION MODEL FlowInAnnulus
end convective flux boundary condition insideWall
```

For free convection heat transfer the far-field temperature can be modeled using a bulk volume element 31.1. This temperature is used in evaluation of film temperature properties needed for calculation of the correlation Grashoff number.

Cases in which the far-field temperature should remain constant (the convective surface is surrounded by an infinite fluid reservoir), are modeled by assigning a large constant volume to the bulk volume element. This will produce solutions in which the bulk volume element temperature will fluctuate while remaining fairly stable near the specified constant temperature value. Cases for which the convective surface is bounded (enclosed) are modeled by assigning the appropriate surrounding volume to the bulk volume element (i.e. the fluid is assumed to be perfectly mixed). Note that in this case the bulk volume element need not exist on the initial discretization/mesh as it can created internally upon startup.

Free convection problems where a variation of temperature is desired requires that the meshed discretization contain a bulk volume element. This bulk volume element will be referenced in one of two ways through the bulk volume element command block, matching the name of the command block or with a "use block" command line in the command block. Additionally the mesh must also contain a nodeset for the bulk volume element node. This will enable a direct association of the bulk volume element with a boundary condition that defines the variation of the far-field temperature. An example of this usage is shown below.

```
Begin Temperature Boundary Condition fluid
  add surface nodelist_1   # defined on the bulk volume element node
  Temperature Time function is fluid_temp_time
End

Begin Bulk Fluid Element aBulkNode
  material = aBulk
  Initial Temperature = 288.0
  bulk element volume = constant v = 1.0e4
  bulk eq energy for temperature using p0 with mass
  use block block_2      # block_2 must exist in the mesh
End

begin convective flux boundary condition outsideWall
  add surface surface_2
  use bulk element aBulkNode
  USE CORRELATION CONVECTION MODEL FreeConv
end convective flux boundary condition outsideWall
```

34.1.4 Visualization

Aria provides a means of outputting the heat transfer coefficient via the solution options command block. It must be noted that for a given surface associated with the correlation heat transfer coefficient, such as in a convective boundary condition coupled to an advective bar, only the actual surface can be used for output. This means that an attempt to construct a sideset on an advective bar and output the coefficients on the bar will fail because it is not a part of the associated boundary condition. An example of the syntax is given below:

```
begin solution options
  post process nodal normalized heat_transfer_coefficient on surface_2 as HTCp
end
```

34.2 Heat Transfer Correlation Coefficient

Scope: Equation System

```
Begin Heat Transfer Correlation Coefficient COEFF name
    Apply Gnielinski Film Gradient Correction [ With Exponent {=|are|is} Exponent ]
    Apply Hausen Entrance Effect Correction [ Maximum Correction {=|are|is} MaxCorrection
    ]
    Characteristic Length {=|are|is} Dh
    Compute ComputeCorrParam [ Model {=|are|is} Value...  ]
    Density {=|are|is} rho
    Entrance Length {=|are|is} De
    Expansion Coefficient {=|are|is} beta
    Fluid Phase {=|are|is} FluidPhase
    Friction Factor {=|are|is} f
    Gravitational Constant {=|are|is} g
    Kinematic Viscosity {=|are|is} nu
    Laminar Correlation {=|are|is} Type CorrelationType
    Porosity {=|are|is} phi
    Prandtl Number {=|are|is} Pr
    Reynolds Number {=|are|is} Re
    Turbulent Correlation {=|are|is} Type CorrelationType
    Temperature {=|are|is} t
    Transition Reynolds Number {=|are|is} Rex
    Velocity {=|are|is} v
    Wall Angle {=|are|is} theta
    Wall Length {=|are|is} l
    Wall Temperature {=|are|is} Tw
End
```

Summary — Specified heat transfer correlation coefficient model.

Description — Enables the user to specify a correlation model for use with computing heat transfer coefficients in conjunction with advective bar networks.

34.2.1 Apply Gnielinski Film Gradient Correction

Scope: Heat Transfer Correlation Coefficient

```
Apply Gnielinski Film Gradient Correction [ With Exponent {=|are|is} Exponent ]
```

Parameter	Value	Default
Exponent	real	NONE

Summary Enable a Gnielinski film temperature gradient correction factor to account for variation of fluid properties with temperature when computing the pipe and annular flow correlation heat transfer coefficients (Types 23, 25, 72-74).

Description For gas phase convection the heat transfer coefficient correlations include a correction factor of the form $\left(\frac{T}{T_w}\right)^n$.

For gas cooling $\left(\frac{T}{T_w}\right) > 1.0$ $n = 0$.

For gas heating $0.5 < \left(\frac{T}{T_w}\right) < 1.0$, n varies for different gases but $n = 0.45$ is generally recommended. For CO_2 and steam $n = 0.15$.

For liquids the correction factor is a function of bulk fluid and wall Prandtl number $\left(\frac{Pr_b}{Pr_w}\right)^{0.11}$.

34.2.2 Apply Hausen Entrance Effect Correction

Scope: Heat Transfer Correlation Coefficient

`Apply Hausen Entrance Effect Correction [ Maximum Correction {=|are|is}` *`MaxCorrection`* `]`

Parameter	Value	Default
MaxCorrection	`real`	`10`

Summary Enable the Hausen entrance effect correction factor for pipe and annular flow correlation heat transfer coefficients (Types 23, 25, 72-74).

Description Applies a correction factor of the form $\left(1 + \frac{d_H}{L}^{\frac{2}{3}}\right)$ to the heat transfer coefficient where d_H is the hydraulic diameter and L is the distance from the pipe or annulus entrance to the point the flux is applied. By default the maximum value of the correction is 10, it can optionally be set using the MaxCorrection parameter in this command line. For $L < L_{min}$ the correction factor is not applied. This correction factor may only be applied when the correlation is being used with an advective bar model to represent the fluid flow.

34.2.3 Characteristic Length

Scope: Heat Transfer Correlation Coefficient

`Characteristic Length {=|are|is}` *`Dh`*

Parameter	Value	Default
Dh	`real`	`undefined`

Summary Specifies the characteristic length for parameter computations such as Reynolds Number, etc.

34.2.4 Compute

Scope: Heat Transfer Correlation Coefficient

Compute *ComputeCorrParam* [Model {=|are|is} *Value...*]

Parameter	Value	Default
ComputeCorrParam	{annulus diameter ratio | characteristic length | density | fluid temperature | fluid velocity | friction factor | kinematic viscosity | prandtl number | reynolds number | wall prandtl number | wall temperature}	undefined

Summary Specifies that model parameter be computed through such means as extraction, generic form, specific model, etc. Current options for the parameters and associated models include

CHARACTERISTIC LENGTH: {Hydraulic Diameter | Wetted Perimeter}

FLUID VELOCITY: This should always be set to the name of the advection velocity used in the energy equation. This will most likely be *advection velocity* which is the internal variable used in Aria but may be set manually to anything else such as a user defined velocity field.

FRICTION FACTOR: {Smooth Tube}

34.2.5 Density

Scope: Heat Transfer Correlation Coefficient

Density {=|are|is} *rho*

Parameter	Value	Default
rho	real	undefined

Summary Specify a density to either compute or specify by value for usage with a correlation convection coefficient.

34.2.6 Entrance Length

Scope: Heat Transfer Correlation Coefficient

Entrance Length {=|are|is} *De*

Parameter	Value	Default
De	real	undefined

Summary Specify an entrance length for usage with a correlation convection coefficient.

34.2.7 Expansion Coefficient

Scope: Heat Transfer Correlation Coefficient

Expansion Coefficient {=|are|is} *beta*

Parameter	Value	Default
beta	real	undefined

Summary Specify an expansion coefficient by value for usage with a correlation convection coefficient.

34.2.8 Fluid Phase

Scope: Heat Transfer Correlation Coefficient

`Fluid Phase {= | are | is}` *FluidPhase*

Parameter	Value	Default
FluidPhase	{gas | liquid}	GAS

Summary Specify the fluid phase for usage with a correlation convection coefficient.

34.2.9 Friction Factor

Scope: Heat Transfer Correlation Coefficient

`Friction Factor {= | are | is}` *f*

Parameter	Value	Default
f	real	undefined

Summary Specify a friction factor to either compute or specify by value for usage with a correlation convection coefficient.

34.2.10 Gravitational Constant

Scope: Heat Transfer Correlation Coefficient

`Gravitational Constant {= | are | is}` *g*

Parameter	Value	Default
g	real	undefined

Summary Specify a gravitational constant by value for usage with a correlation convection coefficient.

34.2.11 Kinematic Viscosity

Scope: Heat Transfer Correlation Coefficient

`Kinematic Viscosity {= | are | is}` *nu*

Parameter	Value	Default
nu	real	undefined

Summary Specify a kinematic viscosity to either compute or specify by value for usage with a correlation convection coefficient.

34.2.12 Laminar Correlation

Scope: Heat Transfer Correlation Coefficient

`Laminar Correlation {= | are | is} Type` *`CorrelationType`*

Parameter	Value	Default
CorrelationType	{1 | 14 | 15 | 18 | 2 | 21 | 23 | 25 | 26 | 27 | 28 | 3 | 34 | 35 | 36 | 37 | 38 | 39 | 4 | 5 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 72 | 73 | 74}	undefined

Summary Specifies the laminar correlation model.

34.2.13 Porosity

Scope: Heat Transfer Correlation Coefficient

`Porosity {= | are | is}` *`phi`*

Parameter	Value	Default
phi	real	undefined

Summary Specify a Porosity to either compute or specify by value for usage with a correlation convection coefficient.

34.2.14 Prandtl Number

Scope: Heat Transfer Correlation Coefficient

`Prandtl Number {= | are | is}` *`Pr`*

Parameter	Value	Default
Pr	real	undefined

Summary Specify a Prandtl number to either compute or specify by value for usage with a correlation convection coefficient.

34.2.15 Reynolds Number

Scope: Heat Transfer Correlation Coefficient

`Reynolds Number {= | are | is}` *`Re`*

Parameter	Value	Default
Re	real	undefined

Summary Specify a Reynold's number to either compute or specify by value for usage with a correlation convection coefficient.

34.2.16 Turbulent Correlation

Scope: Heat Transfer Correlation Coefficient

`Turbulent Correlation {= | are | is} Type` *`CorrelationType`*

Parameter	Value	Default
CorrelationType	{1 | 14 | 15 | 18 | 2 | 21 | 23 | 25 | 26 | 27 | 28 | 3 | 34 | 35 | 36 | 37 | 38 | 39 | 4 | 5 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 72 | 73 | 74}	undefined

Summary Specifies the turbulent correlation model.

34.2.17 Temperature

Scope: Heat Transfer Correlation Coefficient

`Temperature {= | are | is}` *t*

Parameter	Value	Default
t	real	undefined

Summary Specify a Fluid Temperature to either compute or specify by value for usage with a correlation convection coefficient.

34.2.18 Transition Reynolds Number

Scope: Heat Transfer Correlation Coefficient

`Transition Reynolds Number {= | are | is}` *Rex*

Parameter	Value	Default
Rex	real	undefined

Summary Specify a transition Reynolds number to either compute or specify by value for usage with a correlation convection coefficient.

34.2.19 Velocity

Scope: Heat Transfer Correlation Coefficient

`Velocity {= | are | is}` *v*

Parameter	Value	Default
v	real	undefined

Summary Specify a velocity to either compute or specify by value for usage with a correlation convection coefficient.

34.2.20 Wall Angle

Scope: Heat Transfer Correlation Coefficient

`Wall Angle {=|are|is}` *theta*

Parameter	Value	Default
theta	`real`	`undefined`

Summary Specify a wall angle by value for usage with a correlation convection coefficient.

34.2.21 Wall Length

Scope: Heat Transfer Correlation Coefficient

`Wall Length {=|are|is}` *l*

Parameter	Value	Default
l	`real`	`undefined`

Summary Specify a wall length by value for usage with a correlation convection coefficient.

34.2.22 Wall Temperature

Scope: Heat Transfer Correlation Coefficient

`Wall Temperature {=|are|is}` *Tw*

Parameter	Value	Default
Tw	`real`	`undefined`

Summary Specify a Wall Temperature to either compute or specify by value for usage with a correlation convection coefficient.

34.2.23 Use Correlation Convection Model

Scope:

`Use Correlation Convection Model` *Name*

Parameter	Value	Default
Name	`string`	`undefined`

Summary Specifies correlation model for convection coefficient

Chapter 35

Chemistry Overview

Aria provides two ways of modeling chemical contributions to the transport equations it solves: CHEMEQ and General Chemistry. The details of these two approaches are described in the following chapters, and the purpose of this overview is to describe the differences between these approaches and provide guidelines for which one to select for a given application.

35.1 Feature Comparison

As a quick reference, a list of features supported by both solvers is shown in Table 35.1 below.

As the table illustrates, CHEMEQ is best suited for evaluating reactions without species transport in order to obtain a source term for the energy equation. It also links to pressurization zones (described in a later chapter) to enable reduced-cost pressurization calculations. CHEMEQ stores its species concentrations in an element field and updates that element field in time while providing a source term for the energy equation.

On the other hand, General Chemistry is best suited for situations where you want to resolve species transport and reaction. General Chemistry is designed only to provide source terms, so there must be a linear system set up for each DOF (energy, species concentrations, etc...) to actually update its values in time. For cases where the species are not transported, the added cost of setting up this linear system makes CHEMEQ the more efficient choice. One advantage of General Chemistry is the updated reaction input format which allows you to mix different types of reactions in the same mechanism, which is not possible in CHEMEQ unless you implement it yourself in a user subroutine.

Feature	CHEMEQ	General Chemistry
Arrhenius Reactions	X	X
Distributed Arrhenius Reactions	X	X
Pressure Dependent Reactions	X	X
User Subroutine Reactions	X	
Time Dependent Reactions		X
General Form Reactions		X
Prout-Tompkins Reactions	X	X
Can Mix Reaction Types		X
Can be Solved with Species Transport		X
Can be Solved Without Transport	X	
Porous System Reactions		X
Activation Temperature	X	
Deactivation Temperature	X	
Activation Time	X	X
Deactivation Time	X	X
General ODE solver interface	X	X
Arbitrary Concentration Units	X	X
Can link to pressurization zones	X	
Segregated solve	X	X
Coupled solve		X
Provides Energy Source	X	X
Provides Species Sources		X
Reversible Reactions		
Equilibrium Chemistry		

Table 35.1. Feature table for CHEMEQ and General Chemistry.

Chapter 36

CHEMEQ Reference

36.1 Common Input Style

The chapter on general chemistry describes a method for entering reactions in individual blocks. ChemEq also supports this syntax for entering reactions, which looks like

```
Begin Reaction R1
  Reaction is A -> B
  Rate Function = Arrhenius  A = 0.1  Ea = 350  R = 1
  Concentration Function = Standard  mu = 1.0 0.0
  Heat of Reaction = -1e6
End
```

You can enter reactions in ChemEq using the legacy syntax described in this chapter, or the new block syntax. You cannot, however, mix the syntaxes.

36.2 Chemical Heating

Materials undergoing non–diffusive endothermic or exothermic chemical reactions can be modeled in Aria. The effect of such reactions is incorporated into the energy conservation equation as a volumetric heating term, which is calculated from the reaction rates. In this section, we give a brief overview of the equations implemented in Aria[1].

The volumetric heating due to the reactions may be written as

$$\dot{q}(\mathbf{x}, t, T) = \sum_{j=1}^{N_r} \mathrm{R}_j r_j, \tag{36.1}$$

where R_j is the known endothermic or exothermic energy release for step j of the reaction, r_j is the calculated reaction rate for the same step, and the summation over the index j is performed for the N_r reaction steps.

Some of the burden of managing the units properly for this expression fall to the user. As described in a subsequent section, you can specify energy release units in the ChemEq block as either per volume or per unit mass. The units of $\dot{q}$ are always power per unit volume (e.g. $\mathrm{W/m^3}$). When you select the "per unit mass" option, the right hand side of the equation above is multiplied by the local density. For example, if the concentration units are mass fractions, you should use the "per unit mass" option. If the concentration units are mass densities, you should use the "per unit volume" option.

[1] Heat transfer in the presence of chemical reactions is detailed in Glassman's *Combustion* [43].

Step j of a multi–step chemical reaction can be expressed using the stoichiometric equation

$$\sum_{i=1}^{N_s} \nu'_{ij} M_i \rightarrow \sum_{i=1}^{N_s} \nu''_{ij} M_i, \tag{36.2}$$

where M_i is the chemical symbol for species i, ν'_{ij} is the stoichiometric coefficient of the reactant species i in reaction step j, ν''_{ij} is the stoichiometric coefficient of the product species i in reaction step j, and N_s represents the number of chemical species. When a species does not occur as a reactant or product in equation (36.2) for a particular reaction step, the corresponding stoichiometric coefficient is set to zero.

The net rate of change in the concentration of the species is determined from the following ordinary differential equation:

$$\frac{dN_i}{dt} = \sum_{j=1}^{N_r} (\nu''_{ij} - \nu'_{ij}) r_j, \tag{36.3}$$

where N_i is the concentration (or mole fraction) for species i. In equations (36.1) and (36.3), the reaction rates are calculated according to the law of mass action

$$r_j = k_j \prod_{i=1}^{N_s} N_i^{\mu_{ij}} \quad \text{for} \quad j = 1, 2, \ldots, N_r, \tag{36.4}$$

where k_j is the kinetic coefficient for reaction step j, and μ_{ij} is the given concentration exponent for reaction step j and species i. The kinetic coefficient, k_j, for each reaction step is usually determined from the Arrhenius equation, which may be written as

$$k_j = M_j A_j \exp\left(\frac{-E_j}{RT}\right) \left(\frac{p}{p_0}\right)^{\alpha_j} T^{\beta_j}, \tag{36.5}$$

where A_j is the pre-exponential factor, M_j is the rate multiplier, E_j is the activation energy, R is the appropriate universal gas constant, T is the temperature, β_j is the steric coefficient, p is the pressure, p_0 is a reference pressure and α_j is the pressure exponent factor. Upon substitution of (36.5) and (36.4) into (36.3), we obtain a nonlinear system of ordinary differential equations for the evolution of the chemical species concentrations, namely

$$\frac{dN_i}{dt} = \sum_{j=1}^{N_r} \left[(\nu''_{ij} - \nu'_{ij}) A_j M_j \exp\left(\frac{-E_j}{RT}\right) \left(\frac{p}{p_0}\right)^{\alpha_j} T^{\beta_j} \prod_{k=1}^{N_s} N_k^{\mu_{kj}} \right] \tag{36.6}$$

Equation (36.6), with appropriate initial conditions for the species concentrations, must be solved simultaneously with equation (5.9) to obtain the time evolution of the temperature field and species concentrations. This is a difficult coupled system of nonlinear equations to solve, because the time scales associated with the chemical reactions can be very different from the time scale associated with conduction. Furthermore, the time scale associated with one chemical reaction step may be very different from the next. An operator splitting strategy is used to decouple the concentration equations from the energy equation at every time step.

Finally, the thermal properties of the chemical material are regarded as weighted averages of the N_s values associated with each species component. Another weighted average that is sometimes useful when interpreting results is the reacted gas fraction, f_{RG}, which is defined as the fraction of reacting material that exists in gas phase,

$$f_{RG} = \frac{(1 - X_c) \sum_{i=1}^{N_s} N_i g_i}{\sum_{i=1}^{N_s} N_i} \tag{36.7}$$

where X_c represents the condensed fraction for the reactive material, and the parameter g_i is defined as

$$g_i = \begin{cases} 1 & \text{if } N_i \text{ is a gas phase species} \\ 0 & \text{if } N_i \text{ is not a gas phase species} \end{cases} \tag{36.8}$$

36.3 Alternate Reaction Models

The default reaction model, Arrhenius, was described in the previous section. Alternate reaction models can be selected for the ChemEq block, and are described in this section.

36.3.1 Prout-Tompkins Reaction Model

The Prout-Tompkins reaction model is used for a reaction with only one product and one reactant. The concentration of the reactant is noted as α, and must be a normalized quantity (typically mass fraction). The three additional required parameters for the model are m, n, and p. The rate of this reaction is then evaluated as:

$$r_0 = A_0 \exp\left(\frac{-E_0}{RT}\right) \alpha^n \left[1 - \left(1 - 10^{-p}\right) \alpha\right]^m \tag{36.9}$$

The values of m, n, and p are provided to the ChemEq description using a data block.

36.4 Concentration Units

The units of the concentration, N_i, used in the formulation in the previous section are up to the user. They could be molar concentrations, mole fractions, mass fractions, or mass concentrations (densities). For the purpose of evaluating the reactions described in the previous section, this choice does not matter. However, if you are using a pressurization model in conjunction with the ChemEq model, you have to specify which unit system your concentrations are in so the moles of gas produced can be calculated correctly.

You can also extract the concentrations of individual species from ChemEq to expressions, to allow further manipulations like concentration-dependent thermal properties. To use this capability, the units for the ChemEq block must be specified. There is no implicit unit conversion done in this process, so if the ChemEq units are mass fractions, you can only extract them to the mass fraction expressions. Any further conversions required can be done after they are extracted from ChemEq.

36.5 Numerical Solution

Numerical simulations including chemical heating require a coupling of subcycled transient solutions for Equation (36.6) and the energy equation (5.10). Given appropriate initial conditions for the species concentrations and temperatures at each element quadrature point, the strategy employed here is to first solve Equation (36.6) for updated species concentrations and an associated volumetric heating rate. Here solving for values at quadrature points skirts any issues with chemistry averaging for adjacent material blocks. Quadrature point volume heating rates are spatially integrated and applied as a source contribution (5.11) to the energy equation which can then be solved for an updated temperature distribution.

In addition to solving for the chemical chemistry state and heating rates at quadrature points, the CHEMEQ interface returns a characteristic timestep used in its subcycled iteration. Since chemistry is subcycled within the FEM model step this characteristic time step will always be substantially smaller than the time step used for outer iteration. Nevertheless, the chemistry time time step must be considered in the outer iteration time step selection if one is to accurately model the system chemistry. The strategy used here is to enable a user specified scaling of the minimum chemistry time step (via CHEMISTRY STEP MULTIPLIER command) in the time step selection process for the FEM model time step. Selection of the chemistry time step is illustrated schematically in Figure 36.1 below. Note that reduction of the

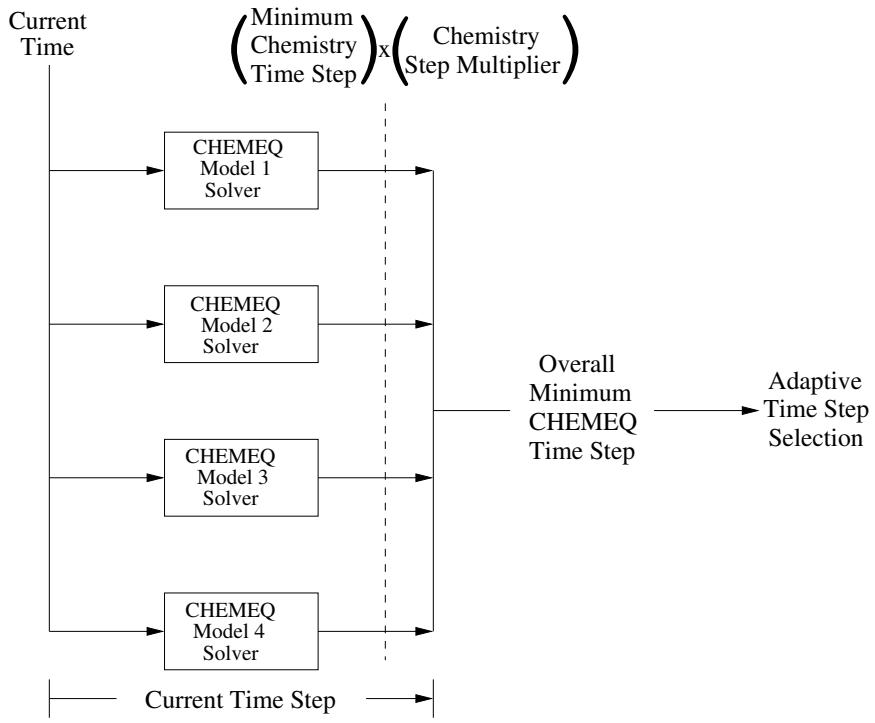


Figure 36.1. Chemistry Time Step Selection

CHEMISTRY STEP MULTIPLIER parameter will often cause the overall minimum chemistry time step to dictate the outer iteration time step.

The numerical model for modeling chemical heating is divided into two sections. One contains a detailed description of the chemical system and the other specifies details of how the set of ordinary differential equations (36.6) are to be solved. Command line input for these two aspects of the numerical model are described in the sections which follow.

The detailed chemical system command line descriptions are provided as an input command block embedded within a higher-level Aria Material command block described in Material Properties (4). It is important to note that chemical heating constitutes a volumetric source term contribution to the energy equation, hence the source must be invoked using a command line as described in Source Terms (11). In most cases chemical heating is thermally activated above some threshold temperature and is controlled independently for each element block to which it is applied. Additional controls for activation and deactivation of the chemical system heat source are provided in the CHEMEQ solver section. The activity status for the source on any element block is available for output as a global variable, *block_number_is_on*.

36.6 Parameters For Chemeq Model

Scope: Material Phase

```
Begin Parameters For Chemeq Model ModelName

    Activation Energies {=|are|is} activation energies...

    Activation Energy St Devs {=|are|is} activation energy st devs...
```

```
    Aux Variable Names {=|are|is} aux variable names...
    Aux Variable Subroutine {=|are|is} aux variable subroutine
    Concentration Exponents For SpeciesName {=|are|is} Exponents...
    Concentration Units {=|are|is} concentration units...
    Condensed Fraction {=|are|is} condensed fraction
    Energy Release Units {=|are|is} energy release units
    Energy Releases {=|are|is} energy releases...
    Enthalpies Of Formation {=|are|is} enthalpies of formation...
    Extent Of Reaction Based On extent of reaction based on...
    Log Preexponential Factors {=|are|is} log preexponential factors...
    Number Of Reactions {=|are|is} number of reactions
    Pressure {=|are|is} pressure...
    Pressure Exponents {=|are|is} pressure exponents...
    Rate Multiplier {=|are|is} rate multiplier...
    Reaction Rate Model {=|are|is} reaction rate model
    Reaction Rate Subroutine {=|are|is} reaction rate subroutine
    Reference Pressure {=|are|is} reference pressure
    Species Molecular Weights {=|are|is} species molecular weights...
    Species Names {=|are|is} species names...
    Species Phases {=|are|is} species phases...
    Steric Coefficients {=|are|is} steric coefficients...
    Stoichiometric Coefficients For SpeciesName {=|are|is} Exponents...
    Begin Reaction ReactionName
    End

End
```

Summary Parameters for the CHEMEQ chemistry model within Aria. The ModelName must appear in a SRC term for an ENERGY equation in order to be applied.

36.6.1 Activation Energies

Scope: Parameters For Chemeq Model

```
Activation Energies {=|are|is} activation energies...
```

Parameter	Value	Default
activation energies	`real...`	`undefined`

Summary This command specifies the activation energies for each reaction.

36.6.2 Activation Energy St Devs

Scope: Parameters For Chemeq Model

`Activation Energy St Devs {=|are|is}` *`activation energy st devs...`*

Parameter	Value	Default
activation energy st devs	`real...`	`undefined`

Summary This command supplies the standard deviations for the activation energies for each reaction. The order is significant, and corresponds to the order in which the Reactions are specified.

36.6.3 Aux Variable Names

Scope: Parameters For Chemeq Model

`Aux Variable Names {=|are|is}` *`aux variable names...`*

Parameter	Value	Default
aux variable names	`string...`	`undefined`

Summary Supplies the names for any auxiliary variables. The number of auxiliary variables is implied by the size of this list.

Description The entries must be separated by whitespace, without commas. These variables do not participate in the reaction kinetics, but are derived from the species variables and calculated in the user-defined subroutine named via the "AUX VARIABLE SUBROUTINE NAME" line command.

36.6.4 Aux Variable Subroutine

Scope: Parameters For Chemeq Model

`Aux Variable Subroutine {=|are|is}` *`aux variable subroutine`*

Parameter	Value	Default
aux variable subroutine	`string`	`undefined`

Summary Supplies the name of the user-defined subroutine that performs any auxiliary variable calculations. This command is required if auxiliary variables are named.

36.6.5 Concentration Exponents For

Scope: Parameters For Chemeq Model

`Concentration Exponents For` *`SpeciesName`* `{=|are|is}` *`Exponents...`*

Parameter	Value	Default
SpeciesName	`string`	`undefined`
Exponents	`real...`	`undefined`

Summary For the given species, this command specifies the concentration exponents for each reaction.

36.6.6 Concentration Units

Scope: Parameters For Chemeq Model

`Concentration Units {=|are|is}` *`concentration units...`*

Parameter	Value	Default
concentration units	`string...`	`UNKNOWN_UNITS`

Summary Specifies the concentration units. This input is only required when using a pressurization model. Valid inputs include: "Mass Fraction", "Mass Fractions", "Mole Fraction", "Mole Fractions", "Mass", "Density", "Moles", or "Molar"

36.6.7 Condensed Fraction

Scope: Parameters For Chemeq Model

`Condensed Fraction {=|are|is}` *`condensed fraction`*

Parameter	Value	Default
condensed fraction	`real`	`undefined`

Summary Specifies mass fraction of the condensed phase.

36.6.8 Energy Release Units

Scope: Parameters For Chemeq Model

Summary Specifies the energy release units for each reaction, PER VOLUME or PER UNIT MASS. Default is PER VOLUME. If PER UNIT MASS is selected, the specified heat release value is multiplied by the local density.

36.6.9 Energy Releases

Scope: Parameters For Chemeq Model

`Energy Releases {=|are|is}` *`energy releases...`*

Parameter	Value	Default
energy releases	`real...`	`undefined`

Summary This command specifies the energy release for each reaction.

36.6.10 Enthalpies Of Formation

Scope: Parameters For Chemeq Model

`Enthalpies Of Formation {=|are|is}` *`enthalpies of formation...`*

Parameter	Value	Default
enthalpies of formation	`real...`	`undefined`

Summary This command specifies the enthalpies of formation for each species

36.6.11 Extent Of Reaction Based On

Scope: Parameters For Chemeq Model

`Extent Of Reaction Based On` *extent of reaction based on...*

Parameter	Value	Default
extent of reaction based on	`string...`	`undefined`

Summary This command supplies the species on which the progress variable for each distributed activation energy depends. The order is significant, and corresponds to the order in which the Reactions are specified.

36.6.12 Log Preexponential Factors

Scope: Parameters For Chemeq Model

`Log Preexponential Factors {=|are|is}` *log preexponential factors...*

Parameter	Value	Default
log preexponential factors	`real...`	`undefined`

Summary This command specifies the natural logarithms of the pre-exponential factors for each reaction.

36.6.13 Number Of Reactions

Scope: Parameters For Chemeq Model

`Number Of Reactions {=|are|is}` *number of reactions*

Parameter	Value	Default
number of reactions	`integer`	`undefined`

Summary Specifies the number of reactions for this material.

36.6.14 Pressure

Scope: Parameters For Chemeq Model

`Pressure {=|are|is}` *pressure...*

Parameter	Value	Default
pressure	`string...`	`undefined`

Summary Specifies the expression for pressure for this material. The preferred option is to use 'Pressure = From_Material_Definition' and put a pressure specification in the material block (e.g. 'Pressure = Pressurization_Model' or 'Pressure = Encore_Function'). If you are using

pressurization zones, you must use this approach. Using 'Pressure = Pressurization_Model' here will generate an error.

36.6.15 Pressure Exponents

Scope: Parameters For Chemeq Model

`Pressure Exponents {=|are|is}` *`pressure exponents...`*

Parameter	Value	Default
pressure exponents	`real...`	`undefined`

Summary This command specifies the pressure exponents for each reaction.

36.6.16 Rate Multiplier

Scope: Parameters For Chemeq Model

`Rate Multiplier {=|are|is}` *`rate multiplier...`*

Parameter	Value	Default
rate multiplier	`string...`	1

Summary Specifies a generic expression for the rate multiplier for this ChemEq model. This input is optional, and the rate multiplier defaults to 1.0 if this is omitted.

36.6.17 Reaction Rate Model

Scope: Parameters For Chemeq Model

`Reaction Rate Model {=|are|is}` *`reaction rate model`*

Parameter	Value	Default
reaction rate model	`string`	`undefined`

Summary By default, reaction rates are Arrhenius. If this line command is included, then a named reaction rate model can be used. Current options are: "Prout_Tompkins"

36.6.18 Reaction Rate Subroutine

Scope: Parameters For Chemeq Model

`Reaction Rate Subroutine {=|are|is}` *`reaction rate subroutine`*

Parameter	Value	Default
reaction rate subroutine	`string`	`undefined`

Summary By default, reaction rates are Arrhenius. If this line command is included, then the named subroutine is called instead.

36.6.19 Reference Pressure

Scope: Parameters For Chemeq Model

`Reference Pressure {=|are|is}` *`reference pressure`*

Parameter	Value	Default
reference pressure	`real`	`undefined`

Summary Specifies the reference pressure for this material.

36.6.20 Species Molecular Weights

Scope: Parameters For Chemeq Model

`Species Molecular Weights {=|are|is}` *`species molecular weights...`*

Parameter	Value	Default
species molecular weights	`real...`	`undefined`

Summary Supplies the molecular weight for each chemical species. The order is significant, and corresponds to the order in which the Species Names are specified.

36.6.21 Species Names

Scope: Parameters For Chemeq Model

`Species Names {=|are|is}` *`species names...`*

Parameter	Value	Default
species names	`string...`	`undefined`

Summary Supplies names for the chemical species. The number of chemical species is implied by the size of this list. The entries must be separated by whitespace, without commas.

36.6.22 Species Phases

Scope: Parameters For Chemeq Model

Summary Supplies the phase for each chemical species. The order is significant, and corresponds to the order in which the Species Names are specified.

36.6.23 Steric Coefficients

Scope: Parameters For Chemeq Model

`Steric Coefficients {=|are|is}` *`steric coefficients...`*

Parameter	Value	Default
steric coefficients	`real...`	`undefined`

Summary This command specifies the steric coefficients for each reaction.

36.6.24 Stoichiometric Coefficients For

Scope: Parameters For Chemeq Model

`Stoichiometric Coefficients For` *SpeciesName* `{=|are|is}` *Exponents...*

Parameter	Value	Default
SpeciesName	string	undefined
Exponents	real...	undefined

Summary For the given species, this command specifies the stoichiometric coefficients for each reaction.

36.7 Chemeq Solver For

Scope: Equation System

`Begin Chemeq Solver For` *ModelName*

`Absolute Tolerance {=|are|is}` *absTol*

`Activation Temperature {=|are|is}` *T_act*

`Activation Temperature Rate {=|are|is}` *T_dot*

`Activation Time {=|are|is}` *T_init*

`Allow Old Heat Generation Method {=|are|is}` *Bool*

`Asymptotic Tolerance {=|are|is}` *Tol*

`Aux Variable` *Name* `{=|are|is}` *C_init*

`Chemistry Step Multiplier {=|are|is}` *Dt_cond_over_dt_chem*

`Deactivation Temperature {=|are|is}` *t ChemistryDeactivationAction*

`Deactivation Temperature Overshoot {=|are|is}` *dT_overshoot*

`Deactivation Temperature Rate {=|are|is}` *T_dot ChemistryDeactivationAction*

`Deactivation Time {=|are|is}` *T_stop ChemistryDeactivationAction*

`Enable Enthalpy Integration`

`Epsilon Max {=|are|is}` *E_max*

`Epsilon Min {=|are|is}` *E_min*

`Maximum Substeps {=|are|is}` *maxSubSteps*

`Minimum Chemistry Timestep {=|are|is}` *Dt_min*

`Minimum Concentration For` *Name* `{=|are|is}` *C_min*

`Ode Solver {=|are|is}` *odeSolver...*

`Percentage Asymptotics {=|are|is}` *PercentValue*

`Post Deactivation Gas Release {=|are|is}` *gasRelease*

`Post Deactivation Heat Release {=|are|is}` *heatRelease*

`Post Deactivation Release Time {=|are|is}` *releaseTime*

`Rate Limiter Factor {=|are|is}` *rateLimiterFactor*

```
    Relative Tolerance {=|are|is} relTol
    Species Name {=|are|is} Conc
End
```

Summary Parameters for the CHEMEQ solver applicable to the previously defined material with CHEMEQ model ModelName. The CHEMEQ SOLVER model produces no independent action since the ModelName must first appear in a SRC term for an ENERGY equation before it be applied.

36.7.1 Absolute Tolerance

Scope: Chemeq Solver For

```
Absolute Tolerance {=|are|is} absTol
```

Parameter	Value	Default
absTol	real	1e-12

Summary Specifies the absolute solution tolerance for the ODE solver

36.7.2 Activation Temperature

Scope: Chemeq Solver For

```
Activation Temperature {=|are|is} T_act
```

Parameter	Value	Default
T_act	real	0.0

Summary Specification of threshold activation temperature for chemistry.

36.7.3 Activation Temperature Rate

Scope: Chemeq Solver For

```
Activation Temperature Rate {=|are|is} T_dot
```

Parameter	Value	Default
T_dot	real	REAL_MAX

Summary Specification of temperature rate limit (T_dot) for which chemistry activation occurs.

36.7.4 Activation Time

Scope: Chemeq Solver For

```
Activation Time {=|are|is} T_init
```

Parameter	Value	Default
T_init	real	REAL_MAX

Summary Specification of activation time for chemistry.

36.7.5 Allow Old Heat Generation Method

Scope: Chemeq Solver For

`Allow Old Heat Generation Method {=|are|is}` *Bool*

Parameter	Value	Default
Bool	{false|true}	FALSE

Summary Specifies whether to allow the use of the old heat generation method, which is known to not satisfy energy conservation.

36.7.6 Asymptotic Tolerance

Scope: Chemeq Solver For

`Asymptotic Tolerance {=|are|is}` *Tol*

Parameter	Value	Default
Tol	real	100.0

Summary Specifies the asymptotic selection criterion for the chemical rate equations.

36.7.7 Aux Variable

Scope: Chemeq Solver For

`Aux Variable` *Name* `{=|are|is}` *C_init*

Parameter	Value	Default
Name	string	undefined
C_init	real	0.0

Summary Specifies the initial concentration for the named auxiliary variable.

36.7.8 Chemistry Step Multiplier

Scope: Chemeq Solver For

`Chemistry Step Multiplier {=|are|is}` *Dt_cond_over_dt_chem*

Parameter	Value	Default
Dt_cond_over_dt_chem	real	100.0

Summary This command specifies the ratio of the maximum allowed conduction timestep to the smallest, current chemistry time step.

36.7.9 Deactivation Temperature

Scope: Chemeq Solver For

`Deactivation Temperature {=|are|is}` *t ChemistryDeactivationAction*

Parameter	Value	Default
t	`real`	`REAL_MAX`
ChemistryDeactivationAction	`{continue|terminate}`	`TERMINATE`

Summary Specification of deactivation temperature and how the simulation should proceed after deactivation.

36.7.10 Deactivation Temperature Overshoot

Scope: Chemeq Solver For

`Deactivation Temperature Overshoot {=|are|is}` *dT_overshoot*

Parameter	Value	Default
dT_overshoot	`real`	`1.0`

Summary Specification of acceptable overshoot of the deactivation temperature for the limiting procedure

36.7.11 Deactivation Temperature Rate

Scope: Chemeq Solver For

`Deactivation Temperature Rate {=|are|is}` *T_dot ChemistryDeactivationAction*

Parameter	Value	Default
T_dot	`real`	`REAL_MAX`
ChemistryDeactivationAction	`{continue|terminate}`	`TERMINATE`

Summary Specification of temperature rate limit (T_dot) for deactivation and how the simulation should proceed after deactivation.

36.7.12 Deactivation Time

Scope: Chemeq Solver For

`Deactivation Time {=|are|is}` *T_stop ChemistryDeactivationAction*

Parameter	Value	Default
T_stop	`real`	`REAL_MAX`
ChemistryDeactivationAction	`{continue|terminate}`	`TERMINATE`

Summary Specification of deactivation time for chemistry and how the simulation should proceed after deactivation.

36.7.13 Enable Enthalpy Integration

Scope: Chemeq Solver For

Summary Enables integration of enthalpy along with species.

36.7.14 Epsilon Max

Scope: Chemeq Solver For

`Epsilon Max {=|are|is}` *E_max*

Parameter	Value	Default
E_max	`real`	`10.0`

Summary Specifies the upper bound convergence criterion and chemistry time scale check.

36.7.15 Epsilon Min

Scope: Chemeq Solver For

`Epsilon Min {=|are|is}` *E_min*

Parameter	Value	Default
E_min	`real`	`0.0001`

Summary Specifies the lower bound convergence criterion and chemistry time scale check.

36.7.16 Maximum Substeps

Scope: Chemeq Solver For

`Maximum Substeps {=|are|is}` *maxSubSteps*

Parameter	Value	Default
maxSubSteps	`integer`	`100000`

Summary Specifies the maximum number of chemistry substeps per global time step.

36.7.17 Minimum Chemistry Timestep

Scope: Chemeq Solver For

`Minimum Chemistry Timestep {=|are|is}` *Dt_min*

Parameter	Value	Default
Dt_min	`real`	`1.0e-15`

Summary Specifies the minimum time step allowed during the solution of the chemical rate equations.

36.7.18 Minimum Concentration For

Scope: Chemeq Solver For

`Minimum Concentration For` *Name* `{=|are|is}` *C_min*

Parameter	Value	Default
Name	`string`	`undefined`
C_min	`real`	`1.0e-12`

Summary Specifies the minimum concentration for the named species. This comes with a caveat for transient calculations with long simulation times. The internal implementation within CHEMEQ will always limit the minimum species concentration which implies that a non-zero value for a reactant will always result in species production. This has implications for mass fraction conservation as these errors eventually grow with long simulation times. Reduction of this value helps to reduce the error but this results in a longer run times because the reaction rate equations become stiffer and this results in a smaller chemistry time step being selected. For a typical foam decomposition simulation, this default value is acceptable and for an explosives simulation, the value may be increased to 1.0e-8.

36.7.19 Ode Solver

Scope: Chemeq Solver For

`Ode Solver {=|are|is}` *odeSolver...*

Parameter	Value	Default
odeSolver	`string...`	`CVODE`

Summary This command selects the ODE solver to use for a segregated solve. For available options, refer to the 'ODE Solvers' section in the 'Chemistry Solver Reference' chapter of the user manual.

36.7.20 Percentage Asymptotics

Scope: Chemeq Solver For

`Percentage Asymptotics {=|are|is}` *PercentValue*

Parameter	Value	Default
PercentValue	`real`	`0.0`

Summary Specifies the percentage of the chemical rate equations that will always be solved using asymptotics.

36.7.21 Post Deactivation Gas Release

Scope: Chemeq Solver For

`Post Deactivation Gas Release {=|are|is}` *gasRelease*

Parameter	Value	Default
gasRelease	`real`	`0.01`

Summary Amount of gas to be released uniformly after deactivation (units are always in moles per volume).

36.7.22 Post Deactivation Heat Release

Scope: Chemeq Solver For

`Post Deactivation Heat Release {=|are|is}` *heatRelease*

Parameter	Value	Default
heatRelease	`real`	`0.01`

Summary Amount of energy to be released uniformly after deactivation. The units must be consistent with the base ChemEq heat release units (energy per volume or energy per mass).

36.7.23 Post Deactivation Release Time

Scope: Chemeq Solver For

`Post Deactivation Release Time {=|are|is}` *releaseTime*

Parameter	Value	Default
releaseTime	`real`	`0.01`

Summary Time after deactivation for constant heat and gas sources to be active

36.7.24 Rate Limiter Factor

Scope: Chemeq Solver For

`Rate Limiter Factor {=|are|is}` *rateLimiterFactor*

Parameter	Value	Default
rateLimiterFactor	`real`	`0.01`

Summary Enables a rate limiter and specifies what fraction of the fluid time step to consider for the limiter time step.

36.7.25 Relative Tolerance

Scope: Chemeq Solver For

`Relative Tolerance {=|are|is}` *relTol*

Parameter	Value	Default
relTol	`real`	`1e-6`

Summary Specifies the relative solution tolerance for the ODE solver

36.7.26 Species

Scope: Chemeq Solver For

`Species` *Name* `{=|are|is}` *Conc*

Parameter	Value	Default
Name	`string`	`undefined`
Conc	`real`	`0.0`

Summary Specifies the initial concentration for the named species.

Chapter 37

General Chemistry Reference

37.1 Introduction

Aria provides a general capability to specify arbitrary chemical mechanisms of irreversible reactions as part of a simulation. Both homogeneous (single phase) and volumetric heterogeneous (multiple material phases) mechanisms are currently supported. The mechanism is defined by the user in the material block in the input deck, and this definition will then provide species, mass, and energy source terms that can be inserted into the various transport equations being solved.

It may be more convenient to specify the mechanism in either a mole-based unit system or a mass-based unit system, so both capabilities are provided. The formulation of both versions of this functionality are described in the following sections.

Reactions can be specified in two ways in a general chemistry block. The first way uses a format similar to CHEMEQ, where all reactions have the same form, and you enter blocks of coefficients. These inputs are described in the following section. The second method is to enter individual reaction blocks, which is described in the Reaction Inputs section later in this chapter. These reaction blocks are placed inside the general chemistry block.

An example input using the traditional input style is

```
BEGIN general chemistry gas_chem
  Species Names = As Bs
  Number of Reactions = 1
  Species Variable Name = density
  Universal Gas Constant = 1.0
  Preexponential Factors = 0.1
  Steric Coefficients    = 0.0
  Activation Energies    = 350.0
  Concentration Exponents for As       =  1.0
  Concentration Exponents for Bs       =  0.0
  Stoichiometric Coefficients for As   = -1.0
  Stoichiometric Coefficients for Bs   =  1.0
  Heats of Reaction                   = -1e6
END
```

and the same mechanism input using reaction blocks would be

```
BEGIN general chemistry gas_chem
  Species Names = As Bs
  Species Variable Name = density
```

```
  Begin Reaction R1
    Reaction is As -> Bs
    Rate Function = Arrhenius  A = 0.1  Ea = 350  R = 1
    Concentration Function = Standard  mu = 1.0 0.0
    Heat of Reaction = -1e6
  End
END
```

These two input methods can also be combined, but the specified number of reactions on the "Number of Reactions" line should be only the number listed in the traditional format, not including reactions inside "Begin Reaction" blocks.

37.2 Required Prerequisites

Using general chemistry requires that several expressions be defined in the same material block or available on the block it is evaluated on. These expressions, and the conditions when they are required, are described in the following list.

- Specific heat of each species (only required if you request a chemistry source term for the energy equation)
- Enthalpy of each species (only required if you request a chemistry source term for the energy equation, if you use an evaluator for specific heat this is created automatically)
- Temperature
- Pressure (optional, if you do not provide it and use a pressure dependent reaction rate it will use 101325 Pa)
- Concentrations (required, you specify what form of concentration, e.g. mass fraction, density, species, or species fraction)
- Molecular weights of each species (required if you use molar concentrations)

The phases for inputs (temperature and pressure) come from NO_MATERIAL_PHASE by default. You can specify a list of temperature phases using the old input syntax to override this default. The pressure, when using the old input syntax always comes from the same phase as temperature. When using the new reaction block format, these can be specified separately for each reaction.

37.3 General Chemistry

Scope: Material Phase

`Begin General Chemistry` *`ModelName`*

`Species Names {=|are|is}` *`species names...`*

`Species Variable Name {=|are|is}` *`species variable name...`*

`Species Phases {=|are|is}` *`species phases...`*

`Number Of Reactions {=|are|is}` *`number of reactions`*

`Universal Gas Constant {=|are|is}` *`universal gas constant`*

```
Preexponential Factors {=|are|is} preexponential factors...
Steric Coefficients {=|are|is} steric coefficients...
Activation Energies {=|are|is} activation energies...
Temperature Phases {=|are|is} temperature phases...
Concentration Exponents For SpeciesName {=|are|is} exponents...
Stoichiometric Coefficients For SpeciesName {=|are|is} coefficients...
Heats of Reaction {=|are|is} heats of reaction...
Enthalpy of Formation For SpeciesName {=|are|is} enthalpy of formation...
Nucleation Time {=|are|is} nucleation time...
Activation Temperature {=|are|is} temperature
Deactivation Temperature {=|are|is} temperature action
Deactivation Temperature Rate {=|are|is} rate action
Deactivation Temperature Overshoot {=|are|is} overshoot
Newton Temperature Solve {=|are|is} Bool
End
```

Summary Parameters for the General Chemistry model within Aria. The ModelName must appear in a SRC term for an ENERGY or MASS equation in order to be applied.

37.3.1 Activation Energies

Scope: General Chemistry

`Activation Energies {=|are|is} activation energies...`

Parameter	Value	Default
activation energies	`real...`	`undefined`

Summary This command specifies the activation energies for each reaction.

37.3.2 Concentration Exponents For

Scope: General Chemistry

`Concentration Exponents For SpeciesName {=|are|is} exponents...`

Parameter	Value	Default
SpeciesName	`string`	`undefined`
Exponents	`real...`	`undefined`

Summary For the given species, this command specifies the concentration exponents for each reaction.

37.3.3 Enthalpy of Formation For

Scope: General Chemistry

`Enthalpy of Formation For` *`SpeciesName`* `{=|are|is}` *`enthalpy of formation`*

Parameter	Value	Default
SpeciesName	`string`	`undefined`
Enthalpy of Formation	`real`	`undefined`

Summary For the given species, this command specifies the enthalpy of formation. This is the preferred method of determining heat release. If this is omitted and the heat of reaction is provided instead, the code attempts to determine a set of formation enthalpies that satisfy the heats of reaction. If this cannot be done, an error is thrown.

37.3.4 Species Variable Name

Scope: General Chemistry

`Species Variable Name {=|are|is}` *`species variable name`*

Parameter	Value	Default
species variable name	`string`	None

Summary Specifies the species variable name. Options currently include "Density" (for mass-based calculations) or "Species" (for molar based calculations).

37.3.5 Heats of Reaction

Scope: General Chemistry

`Heats of Reaction {=|are|is}` *`heats of reaction...`*

Parameter	Value	Default
heats of reaction	`real...`	`0.0`

Summary This command specifies the heat of reaction for each reaction. If the enthalpies of formation are provided, this section is ignored. Otherwise, these are converted to enthalpies of formation as described later in this chapter.

37.3.6 Preexponential Factors

Scope: General Chemistry

`Preexponential Factors {=|are|is}` *`preexponential factors...`*

Parameter	Value	Default
preexponential factors	`real...`	`undefined`

Summary This command specifies the pre-exponential factors for each reaction.

37.3.7 Universal Gas Constant

Scope: General Chemistry

`Universal Gas Constant {=|are|is}` *`universal gas constant`*

Parameter	Value	Default
universal gas constant	`real`	`undefined`

Summary Specifies the universal gas constant to use with any of the traditionally specified reactions. Reactions entered in their own blocks define their own constants.

37.3.8 Nucleation Time

Scope: General Chemistry

`Nucleation Time {=|are|is}` *`nucleation time`*

Parameter	Value	Default
nucleation time	`real`	`undefined`

Summary This is an optional input that specifies a nucleation time to apply to the reactions. If provided, it is applied to all the traditionally specified reactions, but not to reactions entered in their own Reaction block.

37.3.9 Number Of Reactions

Scope: General Chemistry

`Number Of Reactions {=|are|is}` *`number of reactions`*

Parameter	Value	Default
number of reactions	`integer`	`undefined`

Summary Specifies the number of reactions for this material, not including reactions entered in their own Reaction block.

37.3.10 Species Names

Scope: General Chemistry

`Species Names {=|are|is}` *`species names...`*

Parameter	Value	Default
species names	`string...`	`undefined`

Summary Supplies names for the chemical species. The number of chemical species is implied by the size of this list. The entries must be separated by whitespace, without commas.

37.3.11 Species Phases

Scope: General Chemistry

`Species Phases {=|are|is}` *`species phases...`*

Parameter	Value	Default
Species Phases	`string...`	`undefined`

Summary Supplies the phase for each chemical species. The order is significant, and corresponds to the order in which the Species Names are specified.

37.3.12 Temperature Phases

Scope: General Chemistry

`Temperature Phases {=|are|is}` *`temperature phases...`*

Parameter	Value	Default
Temperature Phases	`string...`	`NO_MATERIAL_PHASE`

Summary Supplies the phase for each reaction gets its temperature. This does not apply to reactions entered in Reaction blocks.

37.3.13 Steric Coefficients

Scope: General Chemistry

`Steric Coefficients {=|are|is}` *`steric coefficients...`*

Parameter	Value	Default
steric coefficients	`real...`	`undefined`

Summary This command specifies the steric coefficients for each reaction.

37.3.14 Stoichiometric Coefficients For

Scope: General Chemistry

`Stoichiometric Coefficients For` *`SpeciesName`* `{=|are|is}` *`Exponents...`*

Parameter	Value	Default
SpeciesName	`string`	`undefined`
Exponents	`real...`	`undefined`

Summary For the given species, this command specifies the stoichiometric coefficients for each reaction.

37.3.15 Activation Temperature

Scope: General Chemistry

`Activation Temperature {=|are|is}` *temperature*

Parameter	Value	Default
temperature	`real`	`undefined`

Summary This command specifies the activation temperature for the chemistry mechanism.

37.3.16 Deactivation Temperature

Scope: General Chemistry

`Deactivation Temperature {=|are|is}` *temperature action*

Parameter	Value	Default
temperature	`real`	`undefined`
action	`string`	`undefined`

Summary This command specifies the deactivation temperature for the chemistry mechanism. The action string should be either 'continue' or 'terminate'.

37.3.17 Deactivation Temperature Rate

Scope: General Chemistry

`Deactivation Temperature Rate {=|are|is}` *rate action*

Parameter	Value	Default
rate	`real`	`undefined`
action	`string`	`undefined`

Summary This command specifies the deactivation temperature rate for the chemistry mechanism. The action string should be either 'continue' or 'terminate'.

37.3.18 Deactivation Temperature Overshoot

Scope: General Chemistry

`Deactivation Temperature Overshoot {=|are|is}` *overshoot*

Parameter	Value	Default
overshoot	`real`	`undefined`

Summary This command specifies the allowable overshoot (ΔT) for a temperature deactivation. Failing to stop within $T_{deactivate} + \Delta T$ will result in a failed time step (up to a maximum of three consecutive failures).

37.3.19 Newton Temperature Solve

Scope: General Chemistry

`Newton Temperature Solve {=|are|is} Bool`

Parameter	Value	Default
Bool	`{false|true}`	`false`

Summary This command specifies whether or not Newton's method is desired when solving for the temperature. When set to true, Newton's method is used to calculate the temperature. Otherwise, the temperature is computed assuming a constant heat capacity.

37.4 Molar Formulation of Chemical Mechanism

37.4.1 Species Transport

Consider a multi-step chemical reaction mechanism comprised of j irreversible reactions. This system can be expressed as

$$\sum_{i=1}^{N_s} \bar{\nu}'_{ij} M_i \rightarrow \sum_{i=1}^{N_s} \bar{\nu}''_{ij} M_i, \tag{37.1}$$

where $\bar{\nu}'_{ij}$ is the molar stoichiometric coefficient for reactant species M_i and $\bar{\nu}''_{ij}$ is the molar stoichiometric coefficient for product species M_i, and N_s is the number of chemical species. When a species does not appear in either the reactants or products for a reaction j, its stoichiometric coefficients are set to zero.

The net rate of change of molar concentration of species M_i, summed over all j reactions in the mechanism is then

$$\frac{d[M_i]}{dt} = \sum_{j=1}^{N_r} \left(\bar{\nu}''_{ij} - \bar{\nu}'_{ij}\right) \bar{r}_j, \tag{37.2}$$

where N_r is the number of reactions and $\bar{r}_j$ is the molar reaction rate for reaction j, calculated according to the law of mass action,

$$\bar{r}_j = k_j \prod_{i=1}^{N_s} [M_i]^{\mu_{ij}} \tag{37.3}$$

where μ_{ij} is the concentration exponent for species i in reaction j and $\bar{k}_j$ is the kinetic coefficient for reaction j, modeled using the Arrhenius form

$$\bar{k}_j = \bar{A}_j T^{\beta_j} \exp\left(-\frac{E_j}{R_u T}\right), \tag{37.4}$$

where $\bar{A}_j$ is the pre-exponential factor, β_j is the steric coefficient, E_j is the activation energy, R_u is the universal gas constant, and T is the temperature. Combining all terms, the final expression for the molar production rate of species i is then

$$\frac{d[M_i]}{dt} = \dot{\omega}_i = \sum_{j=1}^{N_r} \left[(\bar{\nu}''_{ij} - \bar{\nu}'_{ij}) \bar{A}_j T^{\beta_j} \exp\left(-\frac{E_j}{R_u T}\right) \prod_{k=1}^{N_s} [M_k]^{\mu_{kj}} \right]. \tag{37.5}$$

It is entirely possible that the simulation might be configured such that the native species solution variable might not be the molar concentration $[M_k]$, but instead might be something like mole fraction, species density, or mass fraction. In these cases, it is up to the user to provide a set of material models to convert whatever the solution variable is into something dimensionally equivalent to molar concentration, and then to provide

the name of the final converted molar concentration variable to the chemistry configuration block in the Aria input deck. Some useful conversion into molar concentration include

$$[M_i] = \frac{\rho X_i}{W} = \frac{\rho Y_i}{W_i} = \frac{\rho_i}{W_i}, \tag{37.6}$$

where X_i is the mole fraction of species i, Y_i is the mass fraction of species i, ρ_i is the density of species i, W_i is the molecular weight of species i, and W is the mixture molecular weight computed as $W = \sum X_i W_i = (\sum Y_i/W_i)^{-1}$.

If the reaction mechanism is specified on a molar basis, then the result of the calculations will be a volumetric mole production rate. If this is to be used by a mass-based species transport equation, then an appropriate source term model should be specified by the user for the species transport equations, that include the requisite unit conversions.

37.4.2 Continuity Transport

Once the individual species source terms have been formulated, the overall continuity equation source term for material phase k can then be computed as a summation over all of the individual species source terms as

$$\frac{d[M]_k}{dt} = \dot{\omega}_{c,k} = \sum_{i=1}^{N_{s,k}} \frac{d[M_i]}{dt} \tag{37.7}$$

where $[M]_k$ is the overall molar concentration in material phase k and the summation is over the $N_{s,k}$ species in material phase k. The volumetric material phase is intended to represent the different states of the material in a single volume, such as can be found in porous flow where there may be a solid skeleton phase, a gas phase, and/or a liquid phase occupying the pores. If there is only a single material phase (homogeneous combustion), then this continuity source term will be zero; it can only be non-zero when mass is being exchanged between material phases.

37.4.3 Energy Transport

The volumetric heating source term presents an interesting complexity, since the form of the source term itself varies depending on the type of energy equation being solved. For example, a user might want to solve an enthalpy equation, a sensible enthalpy equation, or even a temperature equation. The molar enthalpy for a particular species i is defined as

$$\bar{h}_i = \Delta\bar{h}^o_{f,i} + \int_{T_{\text{ref}}}^{T} \bar{c}_{p,i}(T)\, dT \tag{37.8}$$

$$= \Delta\bar{h}^o_{f,i} + \bar{h}_{s,i}, \tag{37.9}$$

where $\Delta\bar{h}^o_{f,i}$ is the molar heat of formation for species i at the reference temperature T_{ref}, $\bar{c}_{p,i}$ is the molar specific heat of species i, and the molar sensible enthalpy $\bar{h}_{s,i}$ is defined as the integral of the specific heat from the reference state up to the temperature of interest. See Section 37.6 for a discussion of specifying standard states for species. The mixture molar enthalpy may be calculated by

$$\bar{h} = \bar{h}_s + \sum_{i=1}^{N_s} \Delta\bar{h}^o_{f,i} X_i, \tag{37.10}$$

where the mixture molar sensible enthalpy is defined as

$$\bar{h}_s = \sum_{i=1}^{N_s} \bar{h}_{s,i} X_i. \tag{37.11}$$

For homogeneous combustion, an enthalpy equation requires no combustion source term at all, but the enthalpy must be known for each individual species, including both the chemical and sensible portions. For poorly-characterized species, the heat of formation might be unknown or harder to measure (or approximate) than the specific heat. In these cases, a sensible enthalpy equation might be more convenient.

The energy source term due to combustion when not using total enthalpy becomes

$$\begin{aligned}
\bar{S}_{h_s} &= -\sum_{i=1}^{N_s} \Delta\bar{h}^o_{f,i}\dot{\omega}_i && (37.12)\\
&= -\sum_{i=1}^{N_s} \Delta\bar{h}^o_{f,i}\left(\sum_{j=1}^{N_r}\left(\bar{\nu}''_{ij}-\bar{\nu}'_{ij}\right)\bar{r}_j\right) && (37.13)\\
&= -\sum_{j=1}^{N_r}\left(\sum_{i=1}^{N_s}\Delta\bar{h}^o_{f,i}\left(\bar{\nu}''_{ij}-\bar{\nu}'_{ij}\right)\right)\bar{r}_j && (37.14)\\
&= -\sum_{j=1}^{N_r}\Delta\bar{h}_{R,j}\bar{r}_j, && (37.15)
\end{aligned}$$

where $\Delta\bar{h}_{R,j}$ is the molar heat of reaction for reaction j, which is potentially easier to measure than the heat of formation for individual species. When specifying the reaction system, either the heat of reaction or the enthalpy of formation can be provided. However, if a set of non-physical heats of reaction is provided, an error will be generated.

Extra care must be taken in the case of volumetric heterogeneous combustion, since mass is conserved globally but not within each individual material phase since mass is converted from one phase to another (e.g. from a porous solid into a gas). In this case, if separate enthalpy transport equations are solved for each material phase, a source term appears in each equation that accounts for the enthalpy addition or removal due to creation or destruction of mass within that phase. See Section 37.7 for further details on the form of this source term.

37.5 Mass Formulation of Chemical Mechanism

37.5.1 Species Transport

The model form presented in the previous section is on a molar basis, but it may be more convenient to formulate the chemical mechanism on a mass basis. In this case, the multi-step chemical reaction mechanism may be written as

$$\sum_{i=1}^{N_s}\nu'_{ij}M_i \rightarrow \sum_{i=1}^{N_s}\nu''_{ij}M_i. \qquad (37.16)$$

Here, the view is taken that the stoichiometric coefficient $\bar{\nu}_{ij}$ is a molar ratio between species i and a reference species, $\bar{\nu}_{ij} = \frac{N_i}{N_{\text{ref}}}$, where the reference species is often taken to be one of the reactants that can be identified as the "fuel". Thus, the mass-based stoichiometric coefficients are $\nu'_{ij} = \bar{\nu}'_{ij}\frac{W_i}{W_{\text{ref}}}$ and $\nu''_{ij} = \bar{\nu}''_{ij}\frac{W_i}{W_{\text{ref}}}$, where W_i is the molecular weight of species i and W_{ref} is the molecular weight of the reference species.

The net rate of change of mass concentration (or species density), is then

$$\begin{aligned}\frac{d\rho_i}{dt} = \dot{\omega}_i''' &= W_i \sum_{j=1}^{N_r} \left(\bar{\nu}_{ij}'' - \bar{\nu}_{ij}'\right) \bar{r}_j \\ &= W_i \sum_{j=1}^{N_r} \frac{W_{\text{ref}}}{W_i} \left(\nu_{ij}'' - \nu_{ij}'\right) \bar{r}_j \\ &= \sum_{j=1}^{N_r} \left(\nu_{ij}'' - \nu_{ij}'\right) r_j, \end{aligned} \tag{37.17}$$

where the mass reaction rate is $r_j = W_{\text{ref}} \bar{r}_j$ and the relations in Equation 37.6 are used to convert molar concentration to species density. The mass reaction rate for reaction j is then specified in terms of mass quantities as

$$\begin{aligned}\bar{r}_j &= \bar{k}_j W_{\text{ref}} \prod_{i=1}^{N_s} \frac{\rho_i^{\mu_{ij}}}{W_i^{\mu_{ij}}} \\ &= k_j \prod_{i=1}^{N_s} \rho_i^{\mu_{ij}} \end{aligned} \tag{37.18}$$

where the mass kinetic coefficient is

$$k_j = \frac{W_{\text{ref}}}{\prod_{i=1}^{N_s} W_i^{\mu_{ij}}} \bar{A}_j T^{\beta_j} \exp\left(-\frac{E_j}{R_u T}\right). \tag{37.19}$$

or, more compactly,

$$k_j = A_j T^{\beta_j} \exp\left(-\frac{E_j}{R_u T}\right), \tag{37.20}$$

where the molecular weights can be "absorbed" into the pre-exponential factor to convert it into a mass-based quantity,

$$A_j = \frac{W_{\text{ref}}}{\prod_{i=1}^{N_s} W_i^{\mu_{ij}}} \bar{A}_j. \tag{37.21}$$

Combining all terms above yields the final expression for the volumetric mass production rate for species i,

$$\frac{d\rho_i}{dt} = \dot{\omega}_i''' = \sum_{j=1}^{N_r} \left[(\nu_{ij}'' - \nu_{ij}') A_j T^{\beta_j} \exp\left(-\frac{E_j}{R_u T}\right) \prod_{k=1}^{N_s} \rho_k^{\mu_{kj}} \right], \tag{37.22}$$

which has the same basic functional form as Equation 37.5, meaning that the same model implementation can be used interchangeably for either a mass-based or a mole-based formulation, as long as all user-provided parameters are in an internally-consistent units basis.

The same caveats exist for the mass formulation as the mole formulation, with regard to proper units for the input species variables and the output species reaction rates. Proper material model and source term models must be specified to convert the desired species solution variable into species densities as an input and to convert the volumetric mass production rate into appropriate units for the transport equation source term. Some common conversions into species density include

$$\rho_i = \rho Y_i = \frac{W_i}{W} \rho X_i = W_i [M_i]. \tag{37.23}$$

37.5.2 Continuity Transport

Once the individual species source terms have been formulated, the overall mass-based continuity equation source term for material phase k can then be computed as a summation over all of the individual species source terms as

$$\frac{d\rho_k}{dt} = \dot{\omega}'''_{c,k} = \sum_{i=1}^{N_{s,k}} \frac{d\rho_i}{dt}, \tag{37.24}$$

where ρ_k is the overall density of material phase k and the summation is over the $N_{s,k}$ species in material phase k. If there is only a single material phase (homogeneous combustion), then this continuity source term will be zero; it can only be non-zero when mass is being exchanged between material phases.

37.5.3 Energy Transport

As with the mole-based formulation, the volumetric heating source term takes on a different form depending on the type of equation being solved. For example, a user might want to solve an enthalpy equation, a sensible enthalpy equation, or even a temperature equation. The enthalpy for a particular species i is defined as

$$h_i = \Delta h^o_{f,i} + \int_{T_{\text{ref}}}^{T} c_{p,i}(T)\, dT \tag{37.25}$$

$$= \Delta h^o_{f,i} + h_{s,i}, \tag{37.26}$$

where $\Delta h^o_{f,i}$ is the mass-based heat of formation for species i at the reference temperature T_{ref}, $c_{p,i}$ is the specific heat of species i, and the sensible enthalpy $h_{s,i}$ is defined as the integral of the specific heat from the reference state up to the temperature of interest. See Section 37.6 for a discussion of specifying standard states for species. The mixture enthalpy may be calculated by

$$h = h_s + \sum_{i=1}^{N_s} \Delta h^o_{f,i} Y_i, \tag{37.27}$$

where the mixture sensible enthalpy is defined as

$$h_s = \sum_{i=1}^{N_s} h_{s,i} Y_i. \tag{37.28}$$

For homogeneous combustion, an enthalpy equation requires no combustion source term at all, but the enthalpy must be known for each individual species, including both the chemical and sensible portions. For poorly-characterized species, the heat of formation might be unknown or harder to measure (or approximate) than the specific heat. In these cases, a sensible enthalpy equation might be more convenient.

For a sensible enthalpy equation, the source term due to combustion becomes

$$
\begin{aligned}
S_{h_s} &= -\sum_{i=1}^{N_s} \Delta h^o_{f,i} \dot{\omega}'''_i && (37.29)\\
&= -\sum_{i=1}^{N_s} \Delta h^o_{f,i} \left(\sum_{j=1}^{N_r} \left(\nu''_{ij} - \nu'_{ij} \right) r_j \right) && (37.30)\\
&= -\sum_{j=1}^{N_r} \left(\sum_{i=1}^{N_s} \Delta h^o_{f,i} \left(\nu''_{ij} - \nu'_{ij} \right) \right) r_j && (37.31)\\
&= -\sum_{j=1}^{N_r} \Delta h_{R,j} r_j, && (37.32)
\end{aligned}
$$

where $\Delta h_{R,j}$ is the mass-based heat of reaction for reaction j, which is potentially easier to measure or approximate than the heat of formation for individual species. When specifying the reaction system, either the heat of reaction or the enthalpy of formation can be provided. However, if a set of non-physical heats of reaction is provided, an error will be generated.

As with the molar formulation, extra care must be taken in the case of volumetric heterogeneous combustion since mass is conserved globally but not within each individual material phase. In this case, if separate enthalpy transport equations are solved for each material phase, a source term appears in each equation that accounts for the enthalpy addition or removal due to creation or destruction of mass within that phase. See Section 37.7 for further details on the treatment of this source term.

37.6 Specifying Standard States for Species Enthalpy

When modeling problems involving chemical reactions with species having different specific heats it is important to specify the standard state for the enthalpy of each species appropriately in order to model the heat release of the reactions correctly. As seen in 37.4.3 and 37.5.3 the enthalpy for a species is given by:

$$
\begin{aligned}
h_i &= \Delta h^o_{f,i} + \int_{T_{\text{ref}}}^{T} c_{p,i}(T)\, dT && (37.33)\\
&= \Delta h^o_{f,i} + h_{s,i}, && (37.34)
\end{aligned}
$$

Each specific heat model in Aria that supports enthalpy evaluation (as required for use with the general chemistry capability) provides two optional parameters for specifying the standard state: "h0" is used to specify $\Delta h^o_{f,i}$, and "T_standard" is used to specify T_{ref}. Both parameters default to 0 if unspecified. In general we recommend following one of two approaches:

1. Use a sensible enthalpy formulation and specify either heats of reaction or species formation enthalpies in the general chemistry block. In this case the "h0" parameter for each species specific heat model should be unspecified so it defaults to 0 and the "T_standard" parameter should be set to the appropriate temperature for the heats of reaction or formation enthalpies you provide.

2. Use a total enthalpy formulation and do not specify any heats of reaction or species formation enthalpies in the general chemistry block. In this case the "h0" parameter for each species should be set to its formation enthalpy and the "T_standard" parameter should be set to the appropriate reference temperature.

37.7 Heterogeneous Reaction Energy Term

As mentioned in 37.4.3 and 37.5.3 volumetric heterogeneous reactions conserve mass overall, but mass is transferred between phases. As a result, when separate enthalpy equations are solved for each phase additional source terms must be added to account for the change of mass within the phase. Two forms of this source are supported in Aria, the default is a total enthalpy conserving form that contributes:

$$S_{h_{dest}} = \dot{\rho_{ij}} h_i(T_{or}) \tag{37.35}$$

$$S_{h_{or}} = -\dot{\rho_{ij}} h_i(T_{or}) \tag{37.36}$$

where *dest* is the phase that species i is in, *or* is the origin phase of the reaction, $\dot{\rho_{ij}}$ is the production (or destruction) rate of species i due to reaction j, and $h_i(T_{or})$ is the enthalpy of species i at the current temperature of the the origin phase. Note that the species and reaction origin phases may be the same, in which case the contributions cancel. This form of the source terms using the enthalpy of the species at the origin phase temperature ensures that no thermodynamically inconsistent behavior (e.g. a hotter destination phase increasing in temperature while the origin phase cools for a reaction with 0 heat of reaction) can occur.

The second form of the source term does not conserve total enthalpy, it maintains constant temperature of both phases (if the specified heat of reaction is 0). This may be activated using the "Phase Transfer Preseved Variable = Temperature" line command in the general chemistry mechanism. This is most useful for chemistry mechanisms where the is insufficient data to construct a complete set of enthalpies of formation and a simple constant heat of reaction is desired. This is achieved by neglecting the balancing contribution of energy to the origin phase of the reaction, Eq. 37.36 and using $h_i(T_{dest})$ in place of $h_i(T_{or})$ for the destination phase contribution.

37.8 Reaction Block Overview

The chemistry module in TFTK provides a common input syntax for specifying chemical reaction mechanisms in both Fuego and Aria. The individual reactions will be entered in reaction blocks that are nested within an overall chemistry block defined by the application.

An example mechanism is shown below, with more details about the available options in the following sections.

```
Begin Reaction R1
  Reaction is A -> B
  Rate Function = Arrhenius  A = 0.1  Ea = 350  R = 1
  Concentration Function = Standard  mu = A 1.0
  Heat of Reaction = -1e6
End

Begin Reaction R2
  Reaction is B -> C
  Rate Function = Arrhenius  A = 0.1  Ea = 350  R = 1
  Concentration Function = Standard  mu = Automatic
  Heat of Reaction = 1e6
End
```

37.9 Reaction Units

The method for specifying units can vary between Fuego and Aria. In Aria, the user specifies the "Species Variable Name". This defines what values the rate functions use for the Y values. There is no implicit conversion, so if the species variable name is DENSITY, the Y values will be in units of mass per volume. Likewise, if the species variable name is SPECIES, the units of Y will be in moles per volume. While it is technically possible to specify a species variable of mass fraction or species fraction (mole fraction), this should be done with caution since unit management will be completely up to the user.

Given the user-specified units of Y, a reaction rate (ω_j) is calculated for each reaction. The time rate of change of a reactant or product in a reaction is simply the product of ω_j with the stoichiometric coefficient for that species. This means if the rate function is kept constant but all stoichiometric coefficients are scaled by a constant factor, the production and consumption rates will also be scaled.

Similarly, the heat source is the product of the calculated ω_j value and the reaction heat release. The resulting units should be in power per volume (e.g. W/m^3). As before, the user should ensure that the units of ω_j and the reaction heat release are consistently defined to produce the appropriate output unit.

37.10 Reaction Inputs

The reactions described above use a common reaction form with an Arrhenius rate and a standard concentration dependence. The chemistry module uses a generic reaction form, where the reaction rate is expressed as as a product of four contributing functions

$$\omega_j = f_m(t,T,P,Y) f_r(t,T,P,Y) f_p(P,T) f_c(Y), \tag{37.37}$$

where f_m is the multiplier function, f_r is the rate function, f_p is the pressure function, and f_c is the concentration function. The available options for each of these functions are outlined in the following section. They can be selected in the input file by adding as many reaction blocks in the parent chemistry description block, as

```
Begin Reaction [REACTION NAME]
  Reaction is A + 1.2B -> 3C
  Multiplier Function = [Optional, See below]
  Rate Function = [Required, See below]
  Pressure Function = [Optional, See below]
  Concentration Function = [Optional, See below]
  Temperature Phase = [GAS_PHASE, SOLID_PHASE, NO_MATERIAL_PHASE (default)]
  Pressure Phase = [GAS_PHASE, SOLID_PHASE, NO_MATERIAL_PHASE (default)]
  Heat of Reaction = [Value]
  Origin Phase Heat Fraction = [Value]
End
```

where each reaction in a given general chemistry block must have a unique name. The reaction string is parsed to determine the appropriate stoichiometry for the reaction. The left hand side and right hand side must be separated by either "->" or "=>". There are multiple options for each of the functions, which are described in the following sections.

If the pressure phase argument is omitted but temperature phase is provided, the pressure phase defaults to the same phase as the temperature. By default, all of the reaction heating is applied in the temperature phase (e.g. solid phase). However, if you specify a value for the "Origin Phase Heat Fraction" between 0 and 1 you can divide that heat between phases. The fraction specified controlls how much energy goes into

the origin phase (a.k.a. temperature phase). The remaining energy is split evenly between the other phases that participate in the chemistry model.

If a reversible reaction is desired, the forward and reverse reaction must be specified in two separate reaction blocks. However, there is an option to say 'Reaction is Reverse of [REACTION NAME]' giving the name of the forward reaction block. This will copy over the reaction string with the opposite signs for the stoichiometric coefficients, the rate function, the temperature and pressure phases, and the equal and opposite heat of reaction. If the forward reaction block contains a concentration function that is not Unity, this statement will also create a Standard concentration function for the reverse reaction with automatic coefficients. These default rate and concentration functions can be overwritten by specifying a new function inside the reverse reaction block. An example of how this option might be used for an equilibrium model is shown below.

```
Begin Reaction [FORWARD REACTION NAME]
  Reaction is A -> B
  Rate Function = [Required, See below]
  Concentration Function = [Optional, See below]
  Heat of Reaction = [Value]
End
Begin Reaction [REVERSE REACTION NAME]
  Reaction is Reverse of [FORWARD REACTION NAME]
  Multiplier Function = [Thermodynamic_Equilibrium, Simplified_Equilibrium, See below] [args]
End
```

A summary of all the reactions parsed from the input is also printed to the log file so you can verify they are evaluating as you have intended. An example of this output is shown below.

```
+-----------------------------------------------------------------------------+
| Reaction Summary                                                            |
+-----------------------------------------------------------------------------+

 Name:                      R1
  Reaction String:          A->B
  Stoich Coeffs:            -1.00000 1.00000
  Temperature Phase:        No Material Phase
  Pressure Phase:           No Material Phase
  Heat of Reaction:         -1.00000e+06
  Multiplier Function:      Unity
  Rate Function:            Arrhenius
      Arguments:            A = 0.100000 Ea = 350.000 beta = 0.00000 R = 1.00000
  Pressure Function:        Unity
  Concentration Function:   Standard
      Arguments:            mu = 1.00000 0.00000
```

37.10.1 Multiplier Functions

Unity

The default multiplier function is unity,

$$f_m(t, T, P, Y) = 1 \tag{37.38}$$

and can be used by either omitting the Multiplier Function line or using

```
Multiplier Function = Unity
```

Tanh_Time

The tanh_time multiplier function uses a hyperbolic tangent function of time to increase the reaction rate at a specified time.

$$f_m(t,T,P,Y) = \frac{1}{2}\left(1+\tanh\left(x_{span}(t-x_{ref})\right)\right) \tag{37.39}$$

The two inputs for this are x_{ref} and x_{span}. If you do not specify an input for the span, it uses $x_{span} = 1/x_{ref}$, unless x_{ref} was set as 0, in which case an error is thrown. Example inputs are shown below.

```
Multiplier Function = Tanh_Time x_ref = 54
Multiplier Function = Tanh_Time x_ref = 2.5 span = 0.1
```

Tanh_Temperature

The tanh_temperature multiplier function uses a hyperbolic tangent function of temperature to increase the reaction rate at a specified temperature.

$$f_m(t,T,P,Y) = \frac{1}{2}\left(1+\tanh\left(x_{span}(T-x_{ref})\right)\right) \tag{37.40}$$

The arguments for this function as the same format as for the tanh_time function.

```
Multiplier Function = Tanh_Temperature x_ref = 450
Multiplier Function = Tanh_Temperature x_ref = 600 span = 10
```

Thermodynamic_Equilibrium

The thermodynamic_equilibrium multiplier function can be used to calculate the inverse equilibrium constant $\frac{1}{K_c}$ using thermodynamic properties. When multiplied by the forward rate function, this will provide a reverse rate function. The equation for K_c satisfies the Van't Hoff equation and uses Gibbs free energy.

$$f_m(t,T,P,Y) = \exp\left\{-\sum_{i=1}^{n_s}\frac{\nu_i}{RT}(TS_i^o - H_i^o)\right\}\left(\frac{RT}{P_{atm}}\right)^{\sum_{i=1}^{n_s}\nu_i} \tag{37.41}$$

In this equation, P_{atm} is the atmospheric pressure, R is the universal gas constant, n_s is the number of species, ν_i is the stoichiometric coefficient of the ith species, and S_i^o and H_i^o are the standard state molar entropy and molar enthalpy of the ith species respectively.

The two optional inputs for this are atmospheric pressure, P_{atm}, and the universal gas constant, R. The default values for these are 101325 Pa and 8.314 J/mol-K respectively. Heat capacity evaluators must be provided with the species properties in order to use this function (the enthalpy and entropy evaluators will be derived from the provided heat capacity evaluators). Example inputs are shown below.

```
Multiplier Function = Thermodynamic_Equilibrium
Multiplier Function = Thermodynamic_Equilibrium  Patm = 101325 R = 8.314
```

Simplified_Equilibrium

The simplified_equilibrium multiplier function can be used to calculate the inverse equilibrium constant $\frac{1}{K_c}$ using a known value of the equilibrium constant (in terms of partial pressures) at a specific temperature and the heat of reaction. When multiplied by the forward rate function, this will provide a reverse rate function. The equation for K_c is the integrated Van't Hoff equation.

$$f_m(t,T,P,Y) = \left(\frac{1}{K_{eq}}\right) \exp\left\{\frac{\Delta H^o}{R}\left(\frac{1}{T} - \frac{1}{T_{eq}}\right)\right\} \left(\frac{RT}{P_{atm}}\right)^{\sum_{i=1}^{n_s} \nu_i} \tag{37.42}$$

In this equation, P_{atm} is the atmospheric pressure, R is the universal gas constant, n_s is the number of species, ν_i is the stoichiometric coefficient of the ith species, K_{eq} is the known equilibrium constant (in terms of partial pressures) at temperature T_{eq}, and ΔH^o is the heat of reaction.

The required input for this is T_{eq}. The optional inputs include K_{eq}, P_{atm}, and R defaulted to 1.0, 101325 Pa, and 8.314 J/mol-K respectively. Either the heat of reaction or the enthalpies of formation must be provided with the reactions in the input file to use this equation accurately. Example inputs are shown below.

```
Multiplier Function = Simplified_Equilibrium  Teq = 800
Multiplier Function = Simplified_Equilibrium  Teq = 800 Keq = 0.6
Multiplier Function = Simplified_Equilibrium  Teq = 800 Keq = 0.6 Patm = 101325 R = 8.314
```

General

The general multiplier function uses a function string supplied in the input file to evaluate the rate generally. Allowable variables in this string include t (time), T (temperature), P (pressure), and any valid species name from the current model. The value used for a species is in whatever unit system the general chemistry model is using. Powers in this string are denoted using the '^' character. These functions can include multi-argument functions like min and max.

An example input for this option is

```
Multiplier Function = General  Function = 0.5*(1+tanh((T-350)^2))*max(T,300)
```

37.10.2 Rate Functions

Unity

The rate function is the only required function, so it has no default like the others. However, you can still select Unity for it if desired, using

```
Rate Function = Unity
```

$$f_r(t,T,P,Y) = 1 \tag{37.43}$$

Arrhenius

The Arrhenius rate function is the most common, and uses

$$f_r(t, T, P, Y) = AT^{\beta} \exp\left(\frac{-E_a}{RT}\right) \tag{37.44}$$

The pre-exponential constant, A, is the only required input. Optional inputs include E_a, β, and R. If omitted, the defaults for these are 0 for E_a and β and 8.314 for R. Some example lines to select this rate function is

```
Rate Function = Arrhenius  A = 100 Ea = 20000
Rate Function = Arrhenius  A = 100 Ea = 350 R = 1
Rate Function = Arrhenius  A = 100 beta = 2 Ea = 350 R = 1
```

Distributed Arrhenius

The Distributed Arrhenius rate function uses a cumulative normal distribution to vary the activation energy with respect to a selected progress species.

$$f_r(t, T, P, Y) = AT^{\beta} \exp\left(\frac{-(E_a + \sigma\zeta)}{RT}\right) \tag{37.45}$$

$$\zeta = CDF^{-1}(1 - Y_j/Y_0) \tag{37.46}$$

where CDF^{-1} is the clipped inverse cumulative normal distribution function, Y_j is the concentration of the selected progress species, Y_0 is the specified reference concentration, and σ is the activation energy variation. In addition to these inputs, the same input rules as for the Arrhenius reaction apply (only A is required of those again). The name specified for the progress species must be a valid species name.

Some example inputs for this rate type are (note that these are shown on multiple lines to fit in the manual, but must be on the same line in your input file):

```
Rate Function = Distributed_Arrhenius  A = 100 Ea = 20000 sigma = 1000 Y0 = 1.2
        ProgressSpecies = N2
Rate Function = Distributed_Arrhenius  A = 100 beta = 2 Ea = 350 R = 1 sigma = 100
        Y0 = 1.2 ProgressSpecies = N2
```

General

The general rate function uses a function string supplied in the input file to evaluate the rate generally. Allowable variables in this string include t (time), T (temperature), P (pressure), and any valid species name from the current model. The value used for a species is in whatever unit system the general chemistry model is using. Powers in this string are denoted using the '^' character. These functions can include multi-argument functions like min and max.

An example input for this option is:

```
Rate Function = General Function = 2.1*exp(-350/T)*N2^3.2*(1-CO2)*max(CO2,0.1)
Rate Function = General Function = (T>300 ? 5.0 : 0.01)*CO2*exp(-350/T)
```

37.10.3 Pressure Functions

Unity

The default pressure function is unity,

$$f_p(P,T) = 1 \tag{37.47}$$

and can be used by either omitting the Pressure Function line or using

```
Pressure Function = Unity
```

Exponential

The exponential pressure function is evaluated as

$$f_p(P,T) = \left(\frac{P}{P_{ref}}\right)^n, \tag{37.48}$$

where P_{ref} must be positive. An example input for this option is

```
Pressure Function = Exponential  P_ref = 1 n = 1.5
```

37.10.4 Concentration Functions

Unity

The default concentration function is unity,

$$f_c(Y) = 1 \tag{37.49}$$

and can be used by either omitting the Concentration Function line or using

```
Concentration Function = Unity
```

Standard

The standard concentration function is evaluated as

$$f_c(Y) = \prod_{i=1}^{N_s} Y_i^{\mu_i} \tag{37.50}$$

The values of μ can be specified manually using a list of entries for μ. This is done by entering a list of name-value pairs for each entry with a non-zero μ value. Alternately, you can provide a list of μ values for all species.

Alternately, if your reaction rate only depends on the concentration of the reactants, and the value of μ is the same as the stoichiometric coefficient, then you can have the values set automatically from the reaction stoichiometry by using the keyword "Automatic".

For example, if your reaction is "2A + B -> C" the automatic option would do the same thing as specifying $\mu = A2B1$. If you have repeated species as both reactants and products, only the reaction portion of the coefficient is considered. For example, "A + B -> 2A + C" would result in automatic exponents of $\mu = A1B1$.

```
Concentration Function = Standard  mu = A 2 B 1
Concentration Function = Standard  mu = 2 1 0
Concentration Function = Standard  mu = Automatic
```

Prout Tompkins

The Prout Tompkins concentration function can only be used with a reaction that has a single reactant and a single product, and is evaluated as

$$f_c(Y) = \alpha^n \left(1 - \left(1 - 10^{-p}\right)\alpha\right)^m \tag{37.51}$$

where α is the concentration of the reactant. An example input for this model is

```
Concentration Function = Prout_Tompkins  n = 1.2 m = 2.5 p = 9
```

Evaporation

The evaporation concentration function is evaluated as

$$f_c(Y) = A\sqrt{\frac{2}{\pi MRT}}\frac{1}{2 - C_c}\left(C_e p_v(T) x_{liq} - C_c p x_{vap}\right) \tag{37.52}$$

$$p_v(T) = p_0 \exp\left(\frac{L}{R}\left(\frac{1}{T_b} - \frac{1}{T}\right)\right) \tag{37.53}$$

where C_c and C_e are the condensation and evaporation coefficients, L is the molar enthalpy of vaporization, T_b is the boiling temperature at p_0, p is the gas phase pressure, A is the surface area, x_{liq} is the mole fraction of the liquid that is evaporating, and x_{vap} is the gas phase mole fraction of the evaporating species. If L is provided in mass units, R must be converted appropriately so their units match.

For reactions that do not include evaporation, species that are defined in the material but not involved in any reactions may be omitted from the chemistry definition block. When using evaporation reactions, this is not allowed since it will alter the value of the mole fractions in the equations above. You must also use a species variable name of 'species' when using evaporation since this requires molar concentrations to get mole fractions.

If you provide only C_c or only C_e, then the coefficients are assumed to be the same.

```
Concentration Function = Evaporation  R=8.314  M=18e-3  Cc=0.01  Tb=373  L=4e4
```

Chapter 38

Chemistry Solver Reference

38.1 Introduction

The chemistry solver block specifies how a given chemistry description is solved. The description can currently only be a general chemistry description, however the ODE solvers can now be selected in a ChemEQ solver block using the same options.

If you do not provide a chemistry solver block for a general chemistry description, it will use the default algorithm (COUPLED).

38.2 Solution Approaches

38.2.1 Coupled Solve

For a coupled solve, the chemistry description provides an instantaneous production rate for each DOF, calculated using values at state NP1. This rate is used directly as a source in the transport equation. This is the default solution algorithm.

This approach provides strong coupling between solution variables, but can limit the fluid time step prohibitively for stiff chemistry sets.

38.2.2 Segregated Solve

The segregated solution uses a frozen flow field to integrate the reaction ODEs from time N to time NP1 and uses the final values to calculate a source term consistent with the selected time integrator for the transport equations. Inputs to the problem, such as pressure, density, or specific heat, are held constant during this integration. The resulting source term is then provided explicitly in the transport equations.

Warning: The segregated chemistry solve is known to give poor results for cases where the reactions are rate-limited by transport. For example, consider a very fast reaction which can fully consume the available reactant in a single fluid time step. At present the segregated solution mechanism will eventually converge to the coupled result with time step refinement for these cases, but will typically require smaller time steps than the coupled solve so it is not recommended for these cases. Future improvements to these cases are possible, please contact sierra-help@sandia.gov if you would like to see them prioritized.

38.3 ODE Solver Parameters

There are several ODE solvers available that can be selected. The basic set of input arguments for an ODE solver are given below, where the specific solver arguments are described in the following subsections.

```
ODE Solver = [SOLVER ARGS]
Absolute Tolerance = 1e-10
Relative Tolerance = 1e-5
Maximum Substeps = 100000
Minimum Step Size = 1e-15
```

38.3.1 CVODE

The CVODE solver is the most flexible of the options available, and often the fastest. It is the only option that allows you to select the method (BDF or ADAMS), the order (max of 5 for BDF and 12 for ADAMS), and the iteration scheme (FUNCTIONAL or NEWTON). For stiff problems, "BDF 5 NEWTON" will be most robust, but is also the most costly. For non-stiff problems, a lower order Adams method, such as "ADAMS 6 FUNCTIONAL" may be better.

The default is "BDF 5 NEWTON" if you only write "ODE Solver = CVODE". A few valid options are shown below.

```
ODE Solver = CVODE [METHOD] [ORDER] [ITERATION\_SCHEME]
ODE Solver = CVODE ADAMS 6 FUNCTIONAL
ODE Solver = CVODE BDF 3 NEWTON
ODE Solver = CVODE
```

38.3.2 LSODE

The LSODE solver can sometimes provide better performance than CVODE for large, stiff problems. It always uses the BDF 5 NEWTON scheme, so there are no extra options. The minimum time in LSODE is set to machine precision and cannot be set by the user.

```
ODE Solver = LSODE
```

38.3.3 RKEH

The RKEH is the Euler-Heun Method. It is a second-order error-adaptive Runge-Kutta scheme. It is typically the slow for stiff systems, but may be advantageous for slowly changing non-stiff systems.

```
ODE Solver = RKEH
```

38.3.4 RK12

The RK12 is the Fehlberg 1-2 Method. It is a second-order error-adaptive Runge-Kutta scheme. It is typically the slow for stiff systems, but may be advantageous for slowly changing non-stiff systems.

```
ODE Solver = RK12
```

38.3.5 RKBS

The RKBS is the Bogacki-Shampine Method. It is a third-order error-adaptive Runge-Kutta scheme. It is typically the slow for stiff systems, but may be advantageous for slowly changing non-stiff systems,

```
ODE Solver = RKBS
```

38.3.6 RKF45

The RKF45 is the Runge-Kutta-Fehlberg 4-5 Method. It is a fourth-order error-adaptive Runge-Kutta scheme. It is typically the slow for stiff systems, but may be advantageous for slowly changing non-stiff systems,

```
ODE Solver = RKF5
```

38.3.7 CashKarp

The CashKarp is the Cash-Karp Method. It is a fourth-order error-adaptive Runge-Kutta scheme. It is typically the slow for stiff systems, but may be advantageous for slowly changing non-stiff systems.

```
ODE Solver = CashKarp
```

38.3.8 DOPRI5

The DOPRI5 is the Dormand-Prince Method. It is a fourth-order error-adaptive Runge-Kutta scheme. It is typically the slow for stiff systems, but may be advantageous for slowly changing non-stiff systems.

```
ODE Solver = DOPRI5
```

38.4 Chemistry Solver Parameters For

Scope: Equation System

```
Begin Chemistry Solver Parameters For ModelName
    Absolute Tolerance {=|are|is} absTol
    Chemistry Solver Algorithm {=|are|is} algorithmType
    Maximum Substeps {=|are|is} maxSubSteps
    Minimum Step Size {=|are|is} minStepSize
    Ode Solver {=|are|is} odeSolver...
    Relative Tolerance {=|are|is} relTol
End
```

Summary Parameters for the chemistry solver applicable to the previously defined material with chemistry model ModelName. The chemistry solver model produces no independent action since the ModelName must first appear in a SRC term for an ENERGY or SPECIES equation before it be applied.

38.4.1 Absolute Tolerance

Scope: Chemistry Solver Parameters For

`Absolute Tolerance {=|are|is}` *absTol*

Parameter	Value	Default
absTol	`real`	`1e-12`

Summary Specifies the absolute solution tolerance for the ODE solver

38.4.2 Chemistry Solver Algorithm

Scope: Chemistry Solver Parameters For

`Chemistry Solver Algorithm {=|are|is}` *algorithmType*

Parameter	Value	Default
algorithmType	`string`	`COUPLED`

Summary This command specifies the coupling algorithm between the chemistry model and transport equations

38.4.3 Maximum Substeps

Scope: Chemistry Solver Parameters For

`Maximum Substeps {=|are|is}` *maxSubSteps*

Parameter	Value	Default
maxSubSteps	`integer`	`100000`

Summary Specifies the maximum number of chemistry substeps per global time step.

38.4.4 Minimum Step Size

Scope: Chemistry Solver Parameters For

`Minimum Step Size {=|are|is}` *minStepSize*

Parameter	Value	Default
minStepSize	`real`	`1e-30`

Summary Specifies the minimum integration step size.

38.4.5 Ode Solver

Scope: Chemistry Solver Parameters For

`Ode Solver {=|are|is}` *`odeSolver...`*

Parameter	Value	Default
odeSolver	`string...`	`CVODE`

Summary This command selects the ODE solver to use for a segregated solve. For available options, refer to the 'ODE Solvers' section of this chapter.

38.4.6 Relative Tolerance

Scope: Chemistry Solver Parameters For

`Relative Tolerance {=|are|is}` *`relTol`*

Parameter	Value	Default
relTol	`real`	`1e-6`

Summary Specifies the relative solution tolerance for the ODE solver

Chapter 39

Pressurization Model Reference

39.1 Pressurization Zones

Beta Capability: The pressurization zones capability is under active development hence this capability should be used with some caution.

Pressure-dependent chemical reactions may produce gas species in a confined space, so the pressure of that confined space must be tracked to resolve the pressure dependency properly. This pressure is often uniform enough that solving the continuity equation is unnecessarily expensive. To accomplish this without continuity, you can specify pressurization zones in the region block in the input file.

For each pressurization zone, you specify a list of source blocks (ChemEQ blocks providing gas to that zone) and pressurization blocks (block in which the gas is accumulating) or pressurized bulk nodes (or both). The available gas volume for the zone is the integration of the gas volume fraction over all the pressurization blocks and bulk nodes you have listed. The temperature of the zone is calculated using a weighted average over the pressurization blocks, which you can select as either a volume average, ideal gas average, mass average, or thermal mass average (ρC_p). The blocks do not have to be contiguous, and the source block does not have to be included in the pressurization blocks.

You can also enter an excess volume in the pressurization zone input block, which is added to the total volume of the blocks and bulk nodes for the pressure calculation. The temperature of this excess volume is assumed to be the same as the one calculated for the listed blocks and bulk nodes unless you specify a different temperature. Depending on what temperature averaging method you select, you may also need to provide a weight to use in conjunction with the specified temperature.

For each pressurization zone, a global variable for pressure, temperature, gas volume, and total moles will be created and shown in the log file. These can be output to heartbeat, included in the Exodus file, and used in Encore or user functions along with other global variables.

39.1.1 Temperature Averaging Methods

This section describes the different averaging methods that can be selected to get an effective temperature (T_{eff}). In all cases, ϕ is the volume fraction of gas. This effective temperature is what is used in the equation of state to calculate the pressure in the pressurization zone.

Volume

When using the volume averaging method, the effective temperature is calculated from

$$T_{eff} = \frac{\int \phi T dV}{\int \phi dV} \tag{39.1}$$

Ideal Gas

When using the ideal gas averaging method, the effective temperature is calculated based on the ideal gas law. Starting with

$$n_{tot} = \frac{PV_{tot}}{RT_{eff}} = \sum n_i = \sum \frac{PV_i}{RT_i} \tag{39.2}$$

By re-arranging these relationships to solve for T_{eff} and converting to integral form we get

$$T_{eff} = \frac{\int \phi dV}{\int \phi / T dV} \tag{39.3}$$

Density

When using the density (ρ) averaging method, the effective temperature is calculated from

$$T_{eff} = \frac{\int \phi \rho T dV}{\int \phi \rho dV} \tag{39.4}$$

Thermal Mass

When using the thermal mass (ρC_p) averaging method, the effective temperature is calculated from

$$T_{eff} = \frac{\int \phi \rho C_p T dV}{\int \phi \rho C_p dV} \tag{39.5}$$

39.1.2 Equations of State

Given the volume, temperature, and number of moles of gas produced by all sources, an appropriate equation of state is still required to calculate pressure. This is specified in the pressurization model block, and can be one of the options listed below. Any arguments required for the equation of state are given on the same line, after its name. All equations of state have the optional gas constant argument, R, which defaults to 8314 J/kmol-K if omitted.

Ideal Gas

The ideal gas equation of state requires no coefficients and evaluates the pressure as

$$p = \frac{nRT}{V}. \tag{39.6}$$

and is specified by "Equation of State = Ideal_Gas" or "Equation of State = Ideal_Gas R = 8.314".

BKWS

The BKWS equation of state requires one parameter, the co-volume (V_{co}). The pressure is then evaluated using:

$$x = \frac{n\kappa V_{co}}{V\sqrt{T+6620}} \tag{39.7}$$

$$Z = 1 + x\exp(0.298x) \tag{39.8}$$

$$p = \frac{ZnRT}{V}. \tag{39.9}$$

It is specified by "Equation of State = BKWS covol = VALUE" or "Equation of State = BKWS covol = VALUE R = 8.314".

Van Der Waal

The VanDerWaal equation of state requires two parameters, a and b. The pressure is then evaluated using:

$$p = \frac{nRT}{V-bn} - a\frac{n^2}{V^2} \tag{39.10}$$

It is specified by "Equation of State = VanDerWaal a = VALUE b = VALUE" or "Equation of State = VanDerWaal a = VALUE b = VALUE R = 8.314".

39.1.3 Venting and Coupling of Multiple Pressurization Zones

As of version 4.40 Aria has some basic capabilities to model venting of a pressurization zone to the environment, as well as to couple the pressure of multiple pressurization zones to one another. Venting may be modeled by setting "VENTING MODEL = VENTED" in the pressurization model block. This also requires specifying a model for the venting flow rate as a function of the pressure in the zone. At present there are 2 supported models, one that accounts for choked flow and one that does not. Both models take the form:

$$\dot{m} = K\rho\sqrt{(P - P_{ambient})/\rho} \tag{39.11}$$

where K is an empirical factor, ρ is the density of the gas in the pressurization zone, P is the pressure in the pressurization zone, and $P_{ambient}$ is the pressure of of the environment the gas is venting to. The choked flow model switches to the form:

$$\dot{m}_{choked} = K\rho\sqrt{P(1 - \frac{1}{P_{crit}})/\rho} \tag{39.12}$$

once $\frac{P}{P_{ambient}} > P_{crit}$ where P_{crit} is the user-specified critical pressure ratio. The value of K can be set using any of the generic material model forms supported by Aria (i.e. constant, polynomial, user function, etc.). A sample vented pressurization zone input deck block is presented below:

```
Begin Pressurization Model pZone1
  Initial Pressure = 13
  Initial Mixture Molecular Weight = 32
  Pressure Unit = psi

  Pressurization Source Blocks = block_1
  Pressurized Blocks = block_1

  Venting Model = Vented
  Venting Model Property Venting_Volumetric_Flow_Rate = K_FACTOR_WITH_CHOKING
      critical_pressure_ratio=2.
  Venting Model Property ambient_pressure = constant value = 13
  Venting Model Property K_factor = constant value=1.e-2

  Equation of State = Ideal_Gas

  Temperature Averaging = rhoCp
End Pressurization Model pZone1
```

Multiple pressurization zones may be connected using the same model forms for the flow rate as are available for venting. This can be used to model multiple sealed volumes connected by a restricted flow path or feature that is initially sealed until it reaches a certain temperature or pressure for example. The coupling is turned on using the syntax:

```
Begin Bulk Node Coupling coupling
  Bulk Nodes = pZone1 pZone2
  Couple density model = k_factor_flow
  Couple species model = k_factor_flow
  additional parameter K_factor = user_function name=k_coupling X=time
End
```

where "pZone1" and "pZone2" are the names of 2 pressurization zones present in the model.

Flow Rate Limiter

When the K_factor value is large and the time step is large, the resulting high flow rate between zones may cause numerical convergence issues. To alleviate this, you can specify a flow rate limiter in your coupling specification, as

```
Begin Bulk Node Coupling coupling12
   Bulk Nodes = PZone1 PZone2
   Couple density model = k_factor_flow flow_rate_limiter=100
   Couple species model = k_factor_flow flow_rate_limiter=100
   additional parameter K_factor = Constant  value = 10
End
```

When using the limiter option, the code first calculates the amount of mass transfer between the two zones that would result in pressure equilibrium. This is divided by the current fluid time step to determine a baseline flow rate. The baseline flow rate is then multiplied by the limiter you specify to obtain the maximum allowable flow rate. In this example, the flow rate is limited to reaching equilibrium in 1 % of the current fluid time step. A larger limiter value results in less clipping of the flow rate (so less limiting). When this limiter is not provided, there is no flow rate limiting.

Coupling Algorithm

When the flow rates between coupled pressurization zones is large, it can cause numerical convergence issues in the overall nonlinear solution. When this is the case, you have the option of using a segregated solution approach for the coupling contributions to the pressurization zones. To use this option, you must first specify an ODE solver block in the Aria region, as shown in the example below. The syntax for this block is the same as for the chemistry ODE solver blocks, and is described in Section 38.3.

```
Begin ODE Solver Parameters PZone
  ODE Solver = CVODE
  Absolute Tolerance = 1e-15
  Relative Tolerance = 1e-9
End ODE Solver Parameters
```

Next, you must select the segregated coupling algorithm in each pressurization zone involved in the coupling network and list which ODE solver to use.

```
Begin Pressurization Model pZone1
  Coupling Algorithm = Segregated
  Solver Name = PZONE
End Pressurization Model pZone1
```

There are a few notes to be aware of when using this option:

- The coupled network will use the ODE solver specified in the first block it stores, so if you specify different solvers in each pressurization model some of them will be ignored.
- The segregated approach only works for CLOSED pressurization zones. If you were using a VENTED pressurization zone in your network to vent to atmosphere, you will need to add a new pressurization zone to represent atmosphere (with a large excess volume and no listed blocks) and vent to that instead.
- For very high flow rates, the flow rate limiter described previously may still be required for good convergence.

39.2 Pressurization Model

Scope: Aria Region

Begin Pressurization Model *ModelName*

Coupling Algorithm {=|are|is} *couplingAlg*

Equation Of State {=|are|is} *equationOfState...*

```
    Excess Volume {=|are|is} excessVolume
    Excess Volume Temperature {=|are|is} excessVolumeTemperature
    Excess Volume Weight {=|are|is} excessVolumeWeight
    Initial Mixture Molecular Weight {=|are|is} initMMW
    Initial Pressure {=|are|is} initPressure
    Pressure Unit {=|are|is} pressureUnit
    Pressurization Source Blocks {=|are|is} sourceBlocks...
    Pressurized Blocks {=|are|is} pressurizedBlocks...
    Pressurized Bulk Nodes {=|are|is} pressurizedBulkNodes...
    Solver Name {=|are|is} solverName
    Temperature Averaging {=|are|is} ventingModel
    Venting Model {=|are|is} ventingModel
    Venting Model Property propertyName {=|are|is} propertyModelAndParams...
End
```

Summary Parameters for the pressurization model

39.2.1 Coupling Algorithm

Scope: Pressurization Model

Summary This command specifies the coupling algorithm (COUPLED or SEGREGATED).

39.2.2 Equation Of State

Scope: Pressurization Model

```
Equation Of State {=|are|is} equationOfState...
```

Parameter	Value	Default
equationOfState	string...	undefined

Summary Specifies the equation of state to use (Ideal_Gas, VanDerWaal, or BKWS). The gas constant is an optional argument (R = value), and defaults to 8314 J/kmol-K if omitted.

39.2.3 Excess Volume

Scope: Pressurization Model

```
Excess Volume {=|are|is} excessVolume
```

Parameter	Value	Default
excessVolume	real	0.0

Summary Specifies the volume of the excess volume. This volume is assumed to be at the average zone temperature unless you also specify an excess volume temperature.

39.2.4 Excess Volume Temperature

Scope: Pressurization Model

`Excess Volume Temperature {=|are|is}` *excessVolumeTemperature*

Parameter	Value	Default
excessVolumeTemperature	`real`	`Copied`

Summary Specifies the temperature of the excess volume. If omitted, the excess volume is treated as the same temperature as the main averaged temperature. If provided, you must also provide a weighting factor to use for RHO and RHO_CP temperature averaging schemes.

39.2.5 Excess Volume Weight

Scope: Pressurization Model

`Excess Volume Weight {=|are|is}` *excessVolumeWeight*

Parameter	Value	Default
excessVolumeWeight	`real`	`1`

Summary Specifies the weighting factor to use for the excess volume. For RHO averaging, this will be the density. For rhoCp averaging, this will be rho times Cp. For volume and ideal gas averaging this should not be provided.

39.2.6 Initial Mixture Molecular Weight

Scope: Pressurization Model

`Initial Mixture Molecular Weight {=|are|is}` *initMMW*

Parameter	Value	Default
initMMW	`real`	`28.97`

Summary Specifies the mixture molecular weight of the initial gas present in the pressurization zone

Description Default is 28.97 g/mol for dry air. This should be specified in the appropriate units for the simulation. If no boundary conditions (e.g. venting or coupling to a gas transport region of the model) are applied to the pressurization zone then the value does not affect the results of the simulation.

39.2.7 Initial Pressure

Scope: Pressurization Model

`Initial Pressure {=|are|is}` *initPressure*

Parameter	Value	Default
initPressure	`real`	`-1.0`

Summary Specifies the initial pressure in the specified pressure units.

39.2.8 Pressure Unit

Scope: Pressurization Model

`Pressure Unit {=|are|is}` *pressureUnit*

Parameter	Value	Default
pressureUnit	`string`	`Pa`

Summary This command specifies the units of pressure that is returned. Options are PA, ATM, and PSI (Default is Pa)

39.2.9 Pressurization Source Blocks

Scope: Pressurization Model

`Pressurization Source Blocks {=|are|is}` *sourceBlocks...*

Parameter	Value	Default
sourceBlocks	`string...`	`undefined`

Summary A list of ChemEq blocks that provide the gas source(s) for this pressurization model.

39.2.10 Pressurized Blocks

Scope: Pressurization Model

`Pressurized Blocks {=|are|is}` *pressurizedBlocks...*

Parameter	Value	Default
pressurizedBlocks	`string...`	`undefined`

Summary A list of blocks that the pressurization occurs in.

39.2.11 Pressurized Bulk Nodes

Scope: Pressurization Model

`Pressurized Bulk Nodes {=|are|is}` *pressurizedBulkNodes...*

Parameter	Value	Default
pressurizedBulkNodes	`string...`	`undefined`

Summary A list of bulk nodes that the pressurization occurs in.

39.2.12 Solver Name

Scope: Pressurization Model

`Solver Name {=|are|is}` *solverName*

Parameter	Value	Default
solverName	`string`	`NONE`

Summary This command specifies the name of the ODE solver block to use for pressurization.

39.2.13 Temperature Averaging

Scope: Pressurization Model

`Temperature Averaging {= | are | is}` *`ventingModel`*

Parameter	Value	Default
ventingModel	`string`	`IDEAL_GAS`

Summary This command specifies the temperature averaging model to use (VOLUME, IDEAL_GAS, RHO, or RHOCP)

39.2.14 Venting Model

Scope: Pressurization Model

`Venting Model {= | are | is}` *`ventingModel`*

Parameter	Value	Default
ventingModel	`string`	`undefined`

Summary This command specifies the venting model to use (OPEN, CLOSED, or VENTED)

39.2.15 Venting Model Property

Scope: Pressurization Model

`Venting Model Property` *`propertyName`* `{= | are | is}` *`propertyModelAndParams...`*

Parameter	Value	Default
propertyName	`string`	`undefined`
propertyModelAndParams	`string...`	`undefined`

Summary Define a material property associated with the venting model. Required if the venting model is VENTED. At a minimum a model for the VENTING_VOLUMETRIC_FLOW_RATE must be provided, and depending on the choice of that model additional properties may be required.

Chapter 40

Burn Front Model Reference

40.1 Burn Front Model

Beta Capability: The burn front model capability is under active development hence this capability should be used with some caution.

Some energetics are modeled in Aria using finite rate chemical mechanisms, however this approach can be more expensive than is necessary to capture some thermal effects. An alternate approach is to define a burn front using a level set, which moves at a prescribed outward velocity and releases heat as it moves.

To enable this capability, the burn front model provides a mechanism for initializing the level set field based on an initiation or ignition temperature.

40.1.1 Model Setup

To set up a burn front model, you must make the following changes to the input file:

1. Define the burn model parameters (burn speed, heat release, initiation, and front width) in each energetic material. The burn speed and heat release can be any generic expression (constant, Encore, etc...).

```
  Burn Speed           = Constant     Value = 0.01   # m/s
  Burn Heat Release    = Constant     Value = 1e6    # J/kg
  Burn Initiation      = Temperature  Tign  = 750    # K
  Burn Front Width     = Constant     Width = 0.005  # m
```

2. Add a level set equation before your temperature equation.

```
Begin Equation System LevelSet
  [SOLVER SETTINGS]

  EQ Level_Set for Level_Set on all_EM Using Q1 with mass ADV
  EQ Extension_Speed for Extension_Speed on all_EM Using Q1 with xfer
End Equation System LevelSet
```

3. Add an energy source term from the level set burn.

```
Source for energy on block_1 = Burn_Front
```

4. Set the initial condition for the level set field to a large positive number everywhere (needs to be larger than the specified level set width).

40.1.2 Model Assumptions and Limitations

This section contains various notes about how the model will behave under certain conditions, and implicit assumptions and limitations of the model.

- By default each block can only have one ignition event. Once it is ignited it initializes the level set field and does not monitor for subsequent ignition. For multiple ignition events, you must specify a 'min_ignition_spacing = X' value on the Burn Initiation line. This will allow multiple ignition events as long as they are separated from an existing burn front by the specified distance. Pick a minimum ignition spacing (d) that is several times larger than your interface width ($d > 3w$), and that provides $Da \gg 1$, where $Da = \frac{v_b d}{\alpha}$ (α = thermal diffusivity and v_b = burn speed). This is so that the thermal front cannot move ahead of the burn front and trigger non-physical internal ignition points.

```
Burn Initiation = Temperature  Tign = 750  min_ignition_spacing = 0.01
```

- The only mechanism for propagation of the burn front between blocks is thermal, where the heat from one material causes another to ignite.
- The level set is distanced using a global distance, so a tortuous material may result in a burn front that jumps from one location to a nearby location. Using the fast-marching level set may alleviate this issue, but is only compatible with tet meshes.
- The total energy release from the burn model is independent of time step size and burn front speed, but the local temperature profile may not be. For example, if a heat flux is applied to a block in a burn front model and a small time step is used, the nodes on the heated surface will ignite and the front will propagate into the block. If a very large time step is used, the entire block may ignite at once releasing its heat uniformly.

Chapter 41

Solution Options

41.1 Solution Options

Scope: Aria Region

```
Begin Solution Options OptionsName

    Apply Flux Limiter Stabilization
    Assemble Tpetra {=|are|is} Value
    Check Matrix For Discrete Maximum Principle
    Face Stabilization Multiplier {=|are|is} Multiplier
    Force Non Tale {=|are|is} Value
    Free Stream Reynolds Number {=|are|is} input_Re
    Ignore Coordinate Displacements {=|are|is} Value
    Maximum Temperature Allowed From Temperature Extraction {=|are|is} MaxTemp
    Maximum Wall Time {=|are|is} WallTime Hours
    Minimum Temperature Allowed From Temperature Extraction {=|are|is} MinTemp
    Omit Enthalpy Adjustment After Temperature Clipping
    Omit Finite Difference Sensitivities For Cantera Properties
    Use Inverse Density Continuity Scaling {=|are|is} Value
    Begin Cvfem Algorithm Specification PStabName
    End

    Begin Edc Model Specification EdcSpecName
    End

    Begin Hdiff Model Specification HdiffOptionsName
    End

    Begin Porous Flow Options blockName
    End

    Begin Species Options blockName
    End

    Begin Turbulence Model Specification TurbSpecName
    End

End
```

Summary Specify information regarding the governing equations to be solved.

41.1.1 Apply Flux Limiter Stabilization

Scope: Solution Options

Summary Enables flux limiter stabilization for diffusion.

Description This activates flux limiter stabilization for diffusion terms to prevent non-physical temperatures on meshes that don't satisfy the requirements for a discrete maximum principle, or for anisotropic material properties.

41.1.2 Assemble Tpetra

Scope: Solution Options

Summary EXPERIMENTAL: Enable (true) to assemble to TPETRA and compare matrix and RHS to FEI matrix.

41.1.3 Check Matrix For Discrete Maximum Principle

Scope: Solution Options

Summary Examine rows of the system matrix before boundary conditions are applied and check if the diagonal entries are non-negative, the off-diagonal entries are non-positive, the row-sum is non-negative. If these conditions fail, it is possible the solution will contain non-physical values, and the discrete maximum principle is not satisfied by the matrix. If any of these conditions fail, a warning is printed to the log file. More output can be obtained with turning on debug logging.

41.1.4 Face Stabilization Multiplier

Scope: Solution Options

`Face Stabilization Multiplier {=|are|is}` *`Multiplier`*

Parameter	Value	Default
Multiplier	`real`	`1.0`

Summary Multiplier for face stabilization term.

Description This value specifies the multiplier applied to the stabilization terms activated by the GRAD_JUMP_PENALT or DMP stabilization terms on the specified equation.

41.1.5 Force Non Tale

Scope: Solution Options

Summary Enable (true) to force simulation not to use TALE.

41.1.6 Free Stream Reynolds Number

Scope: Solution Options

`Free Stream Reynolds Number {=|are|is}` *input_Re*

Parameter	Value	Default
input_Re	`real`	`undefined`

Summary Input Reynolds number is used for solving nondimensional viscous problems using the GasDyn equations.

41.1.7 Ignore Coordinate Displacements

Scope: Solution Options

Summary Enable (true) or disable (false) displacing the physical coordinates by either MESH_DISPLACEMENTS or SOLID_DISPLACEMENTS. By default, this is false, and physical_coordinates are displaced. However, this can be disabled by setting this to true, and physical_coordinates will be equal to model_coordinates.

41.1.8 Maximum Temperature Allowed From Temperature Extraction

Scope: Solution Options

`Maximum Temperature Allowed From Temperature Extraction {=|are|is}` *MaxTemp*

Parameter	Value	Default
MaxTemp	`real`	`2400.0 K`

Summary Specify a maximum cutoff temperature for extraction from enthalpy and composition

Description This option specifies the maximum temperature to be allowed to be extracted from enthalpy, given the mixture composition. If a temperature is computed that is greater than this value, the temperature is reset to equal the maximum allowed value.

41.1.9 Maximum Wall Time

Scope: Solution Options

`Maximum Wall Time {=|are|is}` *WallTime* `Hours`

Parameter	Value	Default
WallTime	`real`	`Infinite`

Summary Specify a maximum wall time to let the simulation end gracefully and output before slurm kills it.

41.1.10 Minimum Temperature Allowed From Temperature Extraction

Scope: Solution Options

`Minimum Temperature Allowed From Temperature Extraction {=|are|is}` *MinTemp*

Parameter	Value	Default
MinTemp	`real`	`250.0 K`

Summary Specify a minimum cutoff temperature for extraction from enthalpy and composition

Description This option specifies the minimum temperature to be allowed to be extracted from enthalpy, given the mixture composition. If a temperature is computed that is less than this value, the temperature is reset to equal the minimum allowed value.

41.1.11 Omit Enthalpy Adjustment After Temperature Clipping

Scope: Solution Options

Summary Do not adjust enthalpy to be consistent with clipped temperatures

Description When temperature extraction fails to produce a viable result, the temperature will usually be clipped at either the upper or lower limit temperature. By default, the enthalpy will then be adjusted up or down to match the clipped temperature so that the thermochemical state is internally consistent. This omits the enthalpy adjustment, leaving clipped nodes in an inconsistent state.

41.1.12 Omit Finite Difference Sensitivities For Cantera Properties

Scope: Solution Options

Summary Do not populate any Cantera property sensitivities via finite difference

41.1.13 Use Inverse Density Continuity Scaling

Scope: Solution Options

Summary Enable (true) or disable (false) the inverse element scaling of the continuity equation.

41.2 Cvfem Algorithm Specification

Scope: Solution Options

`Begin Cvfem Algorithm Specification` *PStabName*

`Activate Acoustic Compressibility Algorithm`

```
    Activate Ausm Plus Scheme
    Activate Edge Based Diffusion Operator [ With CorrectionType ]
    Activate Edge Based Fourth Order Advection
    Activate Kexact Muscl Scheme For Equation EquationString [ {of|species} SpeciesName
    |{in|material_phase} MaterialPhaseName ][ Using K {=|are|is} kappa ]
    Activate Mixture Fraction Clipping Utility At UtilClipLocation
    Activate Muscl Scheme For Equation EquationString [ {of|species} SpeciesName |{in
    |material_phase} MaterialPhaseName ]
    Activate Pressure Projection Algorithm
    Activate Projected Stress Stabilization
    Activate Scv Nodal Gradient
    Activate Shakib Scaling
    Activate Vrtm In Mass Flux Vector
    Ausm Alpha {=|are|is} alpha
    Ausm Beta {=|are|is} beta
    Clip Cvfem Level Set Dof
    First Order Upwind Factor {=|are|is} RealValue For Equation EquationString [ {of
    |species} SpeciesName |{in|material_phase} MaterialPhaseName ]
    Flux Scheme {=|are|is} FluxScheme
    Freeze Muscl Limiter At Global Step {=|are|is} freezeStep
    Hybrid Upwind Factor {=|are|is} RealValue For Equation EquationString [ {of|species}
    SpeciesName |{in|material_phase} MaterialPhaseName ]
    Interpolate Density And Velocity Separately In Mass Flux Vector
    Lag Nodal Pressure Gradient In Mass Flux Vector Expression
    Lag Nodal Tau In Mass Flux Vector Expression
    Muscl Limiter {=|are|is} MusclLimiter For Equation EquationString [ {of|species}
    SpeciesName |{in|material_phase} MaterialPhaseName ]
    Omit Diffusion Term From Sucv Tau
    Omit Disting Bc Ip Sensitivities
    Omit Dt Term From Sucv Tau
    Pressure Stabilization Characteristic Length {=|are|is} CharLength
    Pressure Stabilization Order {=|are|is} Cvfempstaborder
    Pressure Stabilization Parameter Scaling {=|are|is} TauScaling
    Pressure Stabilization Scaling {=|are|is} CvfemPStabScaling [ With Value {=|are|
    is} ConstantTau ]
    Roe Entropy Fix C {=|are|is} epsc
    Roe Entropy Fix U {=|are|is} epsu
    Upwind Method {=|are|is} UpwMethod For Equation EquationString [ {of|species} SpeciesName
    |{in|material_phase} MaterialPhaseName ]
    Use Approximate Roe Sensitivities
    Use Opposing Subface In Open Mass Flux
    Use Specified Pressure In Open Mass Flux
End
```

Summary Specify CVFEM algorithmic modeling options.

41.2.1 Activate Acoustic Compressibility Algorithm

Scope: Cvfem Algorithm Specification

Summary Variable thermodynamic pressure to allow for closed system pressurization

41.2.2 Activate Ausm Plus Scheme

Scope: Cvfem Algorithm Specification

Summary Activate AUSM plus with standard values for alpha and beta

Description Activate AUSM plus scheme.

41.2.3 Activate Edge Based Diffusion Operator

Scope: Cvfem Algorithm Specification

Summary Activate edge-based scheme for diffusion operator.

Description Only implemented now for tke and sdr - interior only

41.2.4 Activate Edge Based Fourth Order Advection

Scope: Cvfem Algorithm Specification

Summary Activate edge-based advection for fourth order LES scheme

Description Activate edge-based advection for fourth order LES scheme

41.2.5 Activate Kexact Muscl Scheme For Equation

Scope: Cvfem Algorithm Specification

Activate Kexact Muscl Scheme For Equation *EquationString* [{of|species} *SpeciesName* |{in|material_phase} *MaterialPhaseName*][Using K {=|are|is} *kappa*]

Parameter	Value	Default
EquationString	string	undefined
SpeciesName	string	undefined
MaterialPhaseName	string	undefined

Summary Use kexact MUSCL variable extrapolation for the specified equation

Description Use kexact MUSCL variable extrapolation for the specified equation. The projected gradients for the necessary variables must be computed using a cvfem_lumped_muscl_projection equation.

41.2.6 Activate Mixture Fraction Clipping Utility At

Scope: Cvfem Algorithm Specification

Summary Activate clipping utility for mixture fraction

Description Sometimes it helps to clip the mixture fraction. This is a utility that provides this support.

41.2.7 Activate Muscl Scheme For Equation

Scope: Cvfem Algorithm Specification

`Activate Muscl Scheme For Equation` *`EquationString`* `[ {of|species}` *`SpeciesName`* `|{in|material_phase}` *`MaterialPhaseName`* `]`

Parameter	Value	Default
EquationString	`string`	`undefined`
SpeciesName	`string`	`undefined`
MaterialPhaseName	`string`	`undefined`

Summary Use MUSCL variable extrapolation for the specified equation

Description Use MUSCL variable extrapolation for the specified equation. The projected gradients for the necessary variables must be computed using a cvfem_lumped_muscl_projection equation.

41.2.8 Activate Pressure Projection Algorithm

Scope: Cvfem Algorithm Specification

Summary Activate PP alg.

Description Rather than fully coupling uvwp, activate a pressure projection algorithm. This will require creating two EQ systems: uvw and p. However, once this is created, the code will handle supplemental utilities. This option can only be used in the context of a momentum lumped mass matrix

41.2.9 Activate Projected Stress Stabilization

Scope: Cvfem Algorithm Specification

Summary Activate full momentum stress term for residual stabilization

Description Activate full momentum stress term for residual stabilization

41.2.10 Activate Scv Nodal Gradient

Scope: Cvfem Algorithm Specification

Summary — Use scv nodal gradient for MUSCL and edge-based diffusion terms

Description — Use scv nodal gradient for MUSCL and edge-based diffusion terms; default is to use nodal grad based on Green Gauss over the duel mesh with edge based integration points.

41.2.11 Activate Shakib Scaling

Scope: Cvfem Algorithm Specification

Summary — Specify Shakib scaling; temp line command

41.2.12 Activate Vrtm In Mass Flux Vector

Scope: Cvfem Algorithm Specification

Summary — Activate velocity relative to mesh in mass flux vector

Description — In some cases, the volume may be constant, although there is mesh motion. This is the case in wind energy sliding mesh applications. This algorithm pays not attention to GCL and assumes that the time rate of volume is zero. Again, this is fine in solid rotation.

41.2.13 Ausm Alpha

Scope: Cvfem Algorithm Specification

`Ausm Alpha {=|are|is}` *alpha*

Parameter	Value	Default
alpha	`real`	`0.0`

Summary — Set constant alpha in Mach splitting function.

Description — Default is 0, unless ausm+ is chosen, then the default is 3/16.

41.2.14 Ausm Beta

Scope: Cvfem Algorithm Specification

`Ausm Beta {=|are|is}` *beta*

Parameter	Value	Default
beta	`real`	`0.0`

Summary Set constant beta in Mach splitting function.

Description Default is 0, unless ausm+ is chosen, then the default is 1/8..

41.2.15 Clip Cvfem Level Set Dof

Scope: Cvfem Algorithm Specification

Summary Clip level set; only germane for conserved level set formulation that has bounds between zero and unity

41.2.16 First Order Upwind Factor

Scope: Cvfem Algorithm Specification

`First Order Upwind Factor {=|are|is}` *`RealValue`* `For Equation` *`EquationString`* `[ {of|species}` *`SpeciesName`* `|{in|material_phase}` *`MaterialPhaseName`* `]`

Parameter	Value	Default
RealValue	`real`	`1.0`
EquationString	`string`	`undefined`
SpeciesName	`string`	`undefined`
MaterialPhaseName	`string`	`undefined`

Summary First-order upwind factor, $0 < x \leq 1$.

Description This value specifies the explicit upwind blending between pure upwind and the chosen convection operator, e.g., UPW*(firstOrderUpwind) + (1-firstOrderUpwind)*(blendedUpwindCentral).

where UPW is the pure first order upwind value and blendedUpwindCentral is a blend between the selected upwind method and central difference operator based on the local cell Peclet number (see Hybrid Upwind Factor line command).

For now, the blendedUpwindCentral is pure central.

41.2.17 Flux Scheme

Scope: Cvfem Algorithm Specification

Summary Set flux scheme used by all equations in the high-Mach CVFEM formulation.

Description Set flux scheme used by all equations in the high-Mach CVFEM formulation.

41.2.18 Freeze Muscl Limiter At Global Step

Scope: Cvfem Algorithm Specification

`Freeze Muscl Limiter At Global Step {=|are|is}` *`freezeStep`*

Parameter	Value	Default
freezeStep	`real`	`undefined`

Summary Freeze MUSCL limiter at a given global step

Description Freeze the MUSCL limiter

41.2.19 Hybrid Upwind Factor

Scope: Cvfem Algorithm Specification

`Hybrid Upwind Factor {=|are|is}` *`RealValue`* `For Equation` *`EquationString`* `[ {of|species}` *`SpeciesName`* `|{in|material_phase}` *`MaterialPhaseName`* `]`

Parameter	Value	Default
RealValue	`real`	`1.0`
EquationString	`string`	`undefined`
SpeciesName	`string`	`undefined`
MaterialPhaseName	`string`	`undefined`

Summary Hybrid upwind factor dials in blending between user specified operator and central.

Description Allows blending of the Peclet factor. A value of zero produces pure central. Higher values blend more user specified upwind (currently, pure first order upwind).

41.2.20 Interpolate Density And Velocity Separately In Mass Flux Vector

Scope: Cvfem Algorithm Specification

Summary use (rho)ip*(uj)ip rather than default (rho*uj)ip

41.2.21 Lag Nodal Pressure Gradient In Mass Flux Vector Expression

Scope: Cvfem Algorithm Specification

Summary Compute the nodal pressure gradient calculation.

Description Lag the nodal pressure gradient assembly for use in the stabilized mass flux vector expression. This command will place the calculation at the top of the non-linear iteration. In general, it will not be wise to use this option for steady simulations as there is one nonlinear loop.

41.2.22 Lag Nodal Tau In Mass Flux Vector Expression

Scope: Cvfem Algorithm Specification

Summary Compute the nodal tau calculation.

Description Lag the nodal tau assembly for use in the stabilized mass flux vector expression. This command will place the calculation at the top of the non-linear iteration. In general, it will not be wise to use this option for steady simulations as there is one nonlinear loop.

41.2.23 Muscl Limiter

Scope: Cvfem Algorithm Specification

`Muscl Limiter {=|are|is}` *`MusclLimiter`* `For Equation` *`EquationString`* `[ {of|species}` *`SpeciesName`* `|{in|material_phase}` *`MaterialPhaseName`* `]`

Parameter	Value	Default
MusclLimiter	`{muscl_minmod|muscl_none|muscl_superbee|muscl_van_albada|muscl_van_leer}`	`undefined`
EquationString	`string`	`undefined`
SpeciesName	`string`	`undefined`
MaterialPhaseName	`string`	`undefined`

Summary Specify the MUSCL limiter method.

Description TVD limiter.

41.2.24 Omit Diffusion Term From Sucv Tau

Scope: Cvfem Algorithm Specification

Summary Do not use timestep in the expression for the SUCV stabilization coefficient

41.2.25 Omit Disting Bc Ip Sensitivities

Scope: Cvfem Algorithm Specification

Summary Drop distinguishing bc sensitivities for ip values.

Description Drop distinguishing bc sensitivities for ip values.

41.2.26 Omit Dt Term From Sucv Tau

Scope: Cvfem Algorithm Specification

Summary Do not use timestep in the expression for the SUCV stabilization coefficient

41.2.27 Pressure Stabilization Characteristic Length

Scope: Cvfem Algorithm Specification

```
Pressure Stabilization Characteristic Length {=|are|is} CharLength
```

Parameter	Value	Default
CharLength	`real`	`undefined`

Summary Specify a constant length scale to compute tau; only applicable for characteristic scaling

41.2.28 Pressure Stabilization Order

Scope: Cvfem Algorithm Specification

```
Pressure Stabilization Order {=|are|is} Cvfempstaborder
```

Parameter	Value	Default
Cvfempstaborder	`{fourth_order|second_order|zeroth_order}`	`undefined`

Summary Order of pressure stabilization

41.2.29 Pressure Stabilization Parameter Scaling

Scope: Cvfem Algorithm Specification

```
Pressure Stabilization Parameter Scaling {=|are|is} TauScaling
```

Parameter	Value	Default
TauScaling	`real`	`undefined`

Summary Scaling of tau

41.2.30 Pressure Stabilization Scaling

Scope: Cvfem Algorithm Specification

```
Pressure Stabilization Scaling {=|are|is} CvfemPStabScaling [ With Value {=|are|is} ConstantTau
]
```

Parameter	Value	Default
CvfemPStabScaling	`{characteristic|constant|shakib|stabilized|time_step}`	`undefined`

Summary Scaling coefficient for pressure stabilization

41.2.31 Roe Entropy Fix C

Scope: Cvfem Algorithm Specification

```
Roe Entropy Fix C {=|are|is} epsc
```

Parameter	Value	Default
epsc	`real`	`0.1`

Summary Small constant for entropy fix (modification of |U +/- c| eigenvalues in Roe scheme)

41.2.32 Roe Entropy Fix U

Scope: Cvfem Algorithm Specification

`Roe Entropy Fix U {= | are | is}` *epsu*

Parameter	Value	Default
epsu	real	0.1

Summary Small constant for entropy fix (modification of |U| eigenvalues in Roe scheme)

41.2.33 Upwind Method

Scope: Cvfem Algorithm Specification

`Upwind Method {= | are | is}` *UpwMethod* `For Equation` *EquationString* `[ {of | species}` *SpeciesName* `| {in | material_phase}` *MaterialPhaseName* `]`

Parameter	Value	Default
UpwMethod	{fourth | gupw | muscl | sucv | upw}	UPW
EquationString	string	undefined
SpeciesName	string	undefined
MaterialPhaseName	string	undefined

Summary Specify method that is blended with pure second order.

Description This user defined method will be blended with pure second order based on cell Peclet blending. In most cases, this operator is upwinded, however, in the case of 4th order a higher order centered scheme is used.

41.2.34 Use Approximate Roe Sensitivities

Scope: Cvfem Algorithm Specification

Summary Use approximate Roe sensitivities for the LHS instead of Van Leer

Description By default, the Roe flux uses the sensitivities for the Van Leer flux scheme form the LHS matrix. This command turns on a different approximation, based more closely on the Roe flux itself; this approximation converges more quickly for some problems, but is less robust for others. Currently this option only has an effect when the "NC_ADV" equation term is used.

41.2.35 Use Opposing Subface In Open Mass Flux

Scope: Cvfem Algorithm Specification

Summary Open mass flux will be computed using the interpolated pressure at the opposing subface

41.2.36 Use Specified Pressure In Open Mass Flux

Scope: Cvfem Algorithm Specification

Summary User specified pressure in open BC will be used to compute gradp in open mass flux BC

41.3 Porous Flow Options

Scope: Solution Options

```
Begin Porous Flow Options blockName
    Integrate Advection By Parts {=|are|is} integrate_advection_by_parts
    Use Cvfem {=|are|is} use_cvfem
End
```

Summary Specify solution options for porous flow.

41.3.1 Integrate Advection By Parts

Scope: Porous Flow Options

Summary Do you wish to integrate the advection associated with porous flow species by parts?

41.3.2 Use Cvfem

Scope: Porous Flow Options

Summary Specify whether to use CVFEM or Galerkin discretization. Default is Galerkin.

41.4 Turbulence Model Specification

Scope: Solution Options

```
Begin Turbulence Model Specification TurbSpecName
    Activate Cvfem Lumped Turbulent Source Term Algorithm
    Activate Kepsilon Source Term Trickster Linearization
    Activate New Komega Src Linearization
    Activate Rhs Sdr Sensitivities To TurbSensParams
    Activate Rhs Tke Sensitivities To TurbSensParams
    Activate Turbulence Clipping Utility At UtilClipLocation
```

```
    Implicit Les Filter Scale {=|are|is} Value
    Limit Turbulent Ke Production To Value Times Dissipation
    Omit Finite Difference Sensitivity Due To Utau
    Omit Sensitivities In Turbulent Production Term
    Omit Sensitivities To Turbulent Production From Ke Source Only
    Omit Velocity Divergence In Turbulent Production Term
    Time Filter {=|are|is} Value
    Turbulence Model {=|are|is} TurbulenceModel
    Turbulence Model Parameter TurbParams {=|are|is} Value
End
```

Summary Specify turbulence modeling options.

41.4.1 Activate Cvfem Lumped Turbulent Source Term Algorithm

Scope: Turbulence Model Specification

Summary Lump cvfem turbulent ke, tdr and ksgs rhs source terms

Description This is appropriate for CVFEM only as it uses a special flavor of lumping source terms.

41.4.2 Activate Kepsilon Source Term Trickster Linearization

Scope: Turbulence Model Specification

Summary Follow classic Patankar approach to linearization of k rhs

Description The dissipation rate in the k equation includes sensitivities to k by using the Prandtl Kolmorgorov relationship to for the model for of the dissipation rate. This procedure provides additional diagonal dominance for the k rhs source term.

41.4.3 Activate New Komega Src Linearization

Scope: Turbulence Model Specification

Summary Alternate komega src linearization

Description Alt komega src linearization

41.4.4 Activate Rhs Sdr Sensitivities To

Scope: Turbulence Model Specification

`Activate Rhs Sdr Sensitivities To` *TurbSensParams*

Parameter	Value	Default
TurbSensParams	{dens_sens | sdr_sens | tke_sens | turb_prod_sens | tvisc_sens}	undefined

---

Summary Dial in rhs sens for sdr equation

41.4.5 Activate Rhs Tke Sensitivities To

Scope: Turbulence Model Specification

`Activate Rhs Tke Sensitivities To` *TurbSensParams*

Parameter	Value	Default
TurbSensParams	{dens_sens | sdr_sens | tke_sens | turb_prod_sens | tvisc_sens}	undefined

---

Summary Dial in rhs sens for tke equation

41.4.6 Activate Turbulence Clipping Utility At

Scope: Turbulence Model Specification

Summary Activate clipping utility for turbulence quantities

Description Sometimes it helps to clip the turbulence quantities. This is a utility that provides this support.

41.4.7 Implicit Les Filter Scale

Scope: Turbulence Model Specification

`Implicit Les Filter Scale {= | are | is}` *Value*

Parameter	Value	Default
Value	real	undefined

Summary Provide implicit filter length

Description In constant coefficient LES models, allow the user to specify and implicit filter length. The default is to determine a length scale tied to the mesh.

41.4.8 Limit Turbulent Ke Production

Scope: Turbulence Model Specification

`Limit Turbulent Ke Production To` *`Value`* `Times Dissipation`

Parameter	Value	Default
Value	`real`	`1.0e-8`

Summary — Choose to limit production source terms

Description — This option limits the turbulent ke production to a scale factor of dissipation, prod = min(prod, limit*den*en1). In practice, the ratio of production to dissipation is not very high. In some flows, it is useful to specify a value of approximately 1000. The ratio should be checked as part of the analysis to make sure that violation of the physical ratio has not been done. In general, this option is only activated in domain locations where dissipation rate is very small.

41.4.9 Omit Finite Difference Sensitivity Due To Utau

Scope: Turbulence Model Specification

Summary — Omit FD sensitivity from utau

Description — The wall friction velocity, utau, can be complex to evaluate in the context of the law of the wall. The default is to compute sensitivities via finite difference. Sometimes, this seems to add some overhead and spurious sensitivities.

41.4.10 Omit Sensitivities In Turbulent Production Term

Scope: Turbulence Model Specification

Summary — Do not include sensitivities for turbulence production

Description — This option removes the analytical sensitivities for the production of turbulent kinetic energy. It seems that most simulations are more stable if this option is activated.

41.4.11 Omit Sensitivities To Turbulent Production From Ke Source Only

Scope: Turbulence Model Specification

Summary — Omit sensitivities to turbulence production from the KE source term

Description — This option removes the analytical sensitivities for the production of turbulent kinetic energy in the KE source terms (where they may be destabilizing) but retains them elsewhere (where they may be helpful). Mutually exclusive w/ the previous option.

41.4.12 Omit Velocity Divergence In Turbulent Production Term

Scope: Turbulence Model Specification

Summary — Do not include divergence term in turbulence production

Description — This option removes the divergence term from the turbulent production of kinetic energy. The default is to include this term.

41.4.13 Time Filter

Scope: Turbulence Model Specification

`Time Filter {= | are | is}` *Value*

Parameter	Value	Default
Value	`real`	`1.0e32`

Summary — Time filter size for Time Filtered Navier-Stokes model

Description — Turbulent viscosity is normally calculated based on a time scale given by k/epsilon (for k-epsilon models) or T (v2f model). The TFNS model substitutes the minimum of the normal computed value and the user-specified time filter size in the turbulent viscosity calculation. In general, this filter should be no less than twice the physical time step. A non-fatal warning is issued if this condition is violated.

41.4.14 Turbulence Model

Scope: Turbulence Model Specification

`Turbulence Model {= | are | is}` *TurbulenceModel*

Parameter	Value	Default
TurbulenceModel	`{dsmag	kepsilon	komega	ksgs	laminar	sarans	smag	sst}`	`undefined`

Summary — Specify type of turbulence model to be used

41.4.15 Turbulence Model Parameter

Scope: Turbulence Model Specification

`Turbulence Model Parameter` *TurbParams* `{= | are | is}` *Value*

Parameter	Value	Default
TurbParams	{a_1 | beta | beta_1 | beta_star | c_epsilon | c_epsilon_1 | c_epsilon_2 | c_mu | c_mu_cs | c_mu_epsilon | edc_c_gamma_lam | edc_c_lam_trans | edc_c_tau_lam | gamma | gamma_1 | kappa | lr_omega | minimum_turbulent_viscosity | pr_t | ramp_time | sa_cb1 | sa_cb2 | sa_cv1 | sa_cw2 | sa_cw3 | sa_sigma | sc_t | sigma_epsilon_lam | sigma_epsilon_trb | sigma_k1_trb | sigma_k_lam | sigma_k_trb | sigma_omega1_trb | sigma_omega_lam | sigma_omega_trb | von_karman | wall_length_factor | y_plus_crit}	undefined
Value	real	undefined

Summary Turbulence model parameters

41.5 Edc Model Specification

Scope: Solution Options

```
Begin Edc Model Specification EdcSpecName

    Activate Co2 Dissociation Model
    Activate Hydrogen Dissociation Model
    Activate Ignition Model {@ | at | for | in | on | over} IgnPart
    Activate Laminar Limit Model
    Activate Lumped Source Term Model
    Activate Separate Co Irreversible Oxidation Pathway
    Activation Time {= | are | is} Time
    Fuel Name {= | are | is} Fuel
    Ignition Threshold Temperature {= | are | is} IgnTemp
    Minimum Product Fraction {= | are | is} Prmin
    Reaction Time Scale {= | are | is} Tchem
    Reference Property {= | are | is} Value
    Reference Property Of Species {= | are | is} Value
    Begin Oxidizer Mixture Specification OxMixName
    End

End
```

Summary Specify EDC combustion model options.

41.5.1 Activate Co2 Dissociation Model

Scope: Edc Model Specification

Summary — Include effects of CO2 dissociation into CO and O2 at high temperatures

Description — At high temperatures, the equilibrium between CO2, CO, and O2 shifts away from CO2, which can significantly decrease the flame temperature. Activating this model will add this effect to the standard EDC combustion model.

41.5.2 Activate Hydrogen Dissociation Model

Scope: Edc Model Specification

Summary — Include effects of H2 dissociation into H

Description — At temperatures greater than about 2000K, the equilibrium between H2 and H will yield non-negligible concentrations of H which can significantly decrease flame temperatures. Activating this model will add this effect to the standard EDC combustion model using the correlations of W.W. Erikson, which are derived from the NASA CEA code [44, 45].

Note that the H species must be included in the Cantera input XML file, and neither H nor H2 should be the "last" species in the list since this species is not independent of the rest (to enforce unity sum) and may be susceptible to more noise than the others. Since temperature and other properties are very sensitive to oscillations in the H and H2 equilibrium, this noise could be problematic.

41.5.3 Activate Ignition Model

Scope: Edc Model Specification

`Activate Ignition Model {@|at|for|in|on|over}` *`IgnPart`*

Parameter	Value	Default
IgnPart	`string`	`undefined`

Summary — MeshPart on which to use the EDC ignition model

Description — Turn on the ignition model for this MeshPart. If fuel and oxidizer are present and the temperature is below the ignition threshold temperature everywhere in the part, reaction will begin. Volume block names and surface names are valid for this command, as well as the alias "all_blocks" and "all_surfaces". Multiple parts must be specified with multiple instances of this line command.

41.5.4 Activate Laminar Limit Model

Scope: Edc Model Specification

Summary — Turn on the EDC laminar limit model for low-turbulence situations

Description Turn on the EDC laminar limit model. This model requires setting three model constants: CtauLam, CgammaLam, and ClamTrans. The model uses a time scale based on a velocity gradient rather than the turb_ke/turb_diss. This appropriate time scale permits the flame to anchor in laminar regions.

41.5.5 Activate Lumped Source Term Model

Scope: Edc Model Specification

Summary Nodally lump the EDC source term

Description If set, the EDC source terms are nodally lumped, rather than using the default implementation of a consistent approach at the Gauss points.

41.5.6 Activate Separate Co Irreversible Oxidation Pathway

Scope: Edc Model Specification

Summary Add CO oxidation pathway as a separate reaction pathway. This should really be used only in the context of a propellant fire in the presence of hydrogen combustion.

41.5.7 Activation Time

Scope: Edc Model Specification

`Activation Time {=|are|is}` *Time*

Parameter	Value	Default
Time	`real`	`0.0`

Summary The time at which the EDC combustion model is activated. No combustion will occur before this time.

41.5.8 Fuel Name

Scope: Edc Model Specification

`Fuel Name {=|are|is}` *Fuel*

Parameter	Value	Default
Fuel	`string`	`undefined`

Summary The name of the EDC fuel species, typically either H2 or a major hydrocarbon

41.5.9 Ignition Threshold Temperature

Scope: Edc Model Specification

`Ignition Threshold Temperature {=|are|is}` *IgnTemp*

Parameter	Value	Default
IgnTemp	`real`	`1000 K`

Summary The temperature below which the ignition model is activated

Description If the ignition model is requested (through the "ACTIVATE IGNITION MODEL" line command, then it will be activated if all temperatures in the corresponding block are below this temperature.

41.5.10 Minimum Product Fraction

Scope: Edc Model Specification

`Minimum Product Fraction {=|are|is}` *Prmin*

Parameter	Value	Default
Prmin	`real`	`1.0e-6`

Summary The minimum product fraction, below which the EDC model will be deactivated.

41.5.11 Reaction Time Scale

Scope: Edc Model Specification

`Reaction Time Scale {=|are|is}` *Tchem*

Parameter	Value	Default
Tchem	`real`	`7.0e-5`

Summary Reaction time scale to set extinction

Description Characteristic time scale of the chemical kinetics. Residence time in the fine structure region will be compared to this to determine if extinction will result.

41.5.12 Reference

Scope: Edc Model Specification

`Reference` *Property* `{=|are|is}` *Value*

Parameter	Value	Default
Property	`string`	`undefined`
Value	`real`	`undefined`

Summary Reference property for initial diagnostic output

Description If all required properties are set, the EDC model will print the adiabatic flame temperature and heat of combustion for the mechanism defined by the specified fuel.

41.5.13 Reference

Scope: Edc Model Specification

```
Reference Property Of Species {=|are|is} Value
```

Parameter	Value	Default
Property	string	undefined
Species	string	undefined
Value	real	undefined

Summary — Reference species-dimensioned property for initial diagnostic output

Description — If all required properties are set, the EDC model will print the adiabatic flame temperature and heat of combustion for the mechanism defined by the specified fuel.

41.6 Oxidizer Mixture Specification

Scope: Edc Model Specification

```
Begin Oxidizer Mixture Specification OxMixName
    Mass_Fraction SpeciesName {=|are|is} Value
    Mole_Fraction SpeciesName {=|are|is} Value
End
```

Summary — Specify either mass fractions or mole fractions for the oxidizer mixture. Only non-zero species need to be included, and the mass or mole fractions must sum to unity. The default is air, with a 0.2095:0.7905 molar ratio between O2 and N2.

41.6.1 Mass_Fraction

Scope: Oxidizer Mixture Specification

```
Mass_Fraction SpeciesName {=|are|is} Value
```

Parameter	Value	Default
SpeciesName	string	undefined
Value	real	undefined

Summary — Oxidizer mixture mass fraction for the given species

41.6.2 Mole_Fraction

Scope: Oxidizer Mixture Specification

```
Mole_Fraction SpeciesName {=|are|is} Value
```

Parameter	Value	Default
SpeciesName	`string`	`undefined`
Value	`real`	`undefined`

Summary Oxidizer mixture mole fraction for the given species

41.6.3 Assemble Tpetra

Scope:

Summary EXPERIMENTAL: Enable (true) to assemble to TPETRA and compare matrix and RHS to FEI matrix.

Chapter 42

Frequently Asked Questions

This chapter includes a compilation of frequently asked questions and problematic scenarios raised by users of the SIERRA Multimechanics Module: Aria. Here the questions are grouped into specific categories of interest but some of the items may appear in more than one category.

42.1 General

Aria example/tutorial files.

SNL internal users can find an Aria training presentation, which includes two PDFs and multiple example files on the following website:
http://compsim.sandia.gov/compsim/
To navigate to the presentations, go to "Support & Services -> Documentation". Once there, go to "Version of the Day -> General Release". This will open a new page with an expandable tree. Go to "Training/Tutorials -> Thermal/Fluid".

Methods for accessing VOTD code.

SNL internal users can access VOTD code on the various platforms by executing `module load sierra/master`.

Use of the VOTD code for production simulations is generally discouraged as stability of the code is subject to daily change. Whenever possible the analyst is encouraged to use the current released distribution executable.

What is the Aria command to specify a log file name.

If the input file is named `base_input.i` then by default the log file is written to `base_input.log`. One can specify an alternative log file by providing optional arguments, either `-o other.log` or `-l other.log` to the job execution command line.

What optional arguments can be used with the Aria command.

The list of arguments which can be used with the Aria command can be listed by:

```
aria -h
```

Is the `includefile` command supported in Aria?

Yes, this feature is supported in all Sierra Mechanics applications. However, the aprepro `include` command is more robust and supports nested include files, while Sierra's `includefile` command does not.

What is the difference between History and Heartbeat output?

The big difference between history and heartbeat output is the type of output. History output is binary format, like Exodus, while Heartbeat output is in ASCII format, like a .txt file. Thus the History output must be visualized with a graphical tool while Heartbeat output is simply inspected or used to generate x-y plots.

Why does restart output a file per processor?

The reason for this is that one typically continues a restart on the same number of processors as the original run. This is also the reason why restart files aren't concatenated at the end of a successful run. Additionally, unless the intent is to overwrite a restart file, one must specify different names for the input and output restart files in the Restart Data block.

Recently the ability to restart with a different number of processors has been provided but requires that one first concatenate the original files and perform the decomposition on the new processor count.

Can one include a text file in an input deck?

Yes, Aprepro provides a way to do this. The syntax is as follows:

```
{include "materials.dat"}
```

Aprepro must be executed on the input deck before running.

42.2 Capability

Can one use steady-state simulation results to specify initial conditions on a different geometry.
There are two possible approaches to accomplish this:

1. using the `omit block` option with the initial conditions via the IC read file option
2. Interpolate the simulation results onto the new geometry using transfers from the Input_Output Region.

Can Aria solve a condensation problem?
There is ongoing work in studies of evaporation, but condensation is not currently supported.

Can one use a FORTRAN user subroutine for heat flux boundary condition?
Yes one can do this but FORTRAN subroutines are currently available only for heat transfer problems.

What can one model with shell elements in Aria.
Typical usage of thermal shell elements in Aria is for materials, which have no through-thickness temperature variation but have thermal diffusion in the plane of the shell. The shell can also account for thermal capacitance. Cases where the through thickness resistance of heat flow through the material is more important will generally be modeled with resistance contact (no shell). The in-between case is a combination of the above. Limited support is also available for shells with through-thickness temperature variation but these models are restricted to having only shell elements.

Does Aria support 1D elements, such as bars?
Aria does support 2D and 3D bar elements however no provision is made for application of flux boundary conditions at the end of the bar. Thus in many cases one may be better off using volume elements instead.

How does one implement a volumetric source on a 2D mesh.
The fundamental assumption in 2D FEM Cartesian formulations is that of a unit depth cross-section in defining the corresponding volume. Hence, the command `add volume block_name` will work in both 2D and 3D cases.

Can Aria be run explicitly?
Explicit methods do circumvent the complicated details of efficient preconditioners and general solver efficiencies based on both matrix condition number and potential high core counts with generally low work per core. However, depending on the time scales of interest of the problem, explicit methods can be painfully inefficient. In general, production Aria does not support explicit methods.

How to load a single displacement field.
To load a previously computed displacement field and use it throughout the Aria thermal simulation, one needs to extract the displacement solution from a single time plane in the displacements file and assign it to the `mesh_displacements` Field in Aria. Internally Aria will update the current configuration, `model_coordinates + mesh_displacements`, and compute on that geometry.

Is it possible to write a plugin to create a boundary condition that prescribes an angular velocity to a block?
Yes, one can write plugins for boundary conditions. However, in order to apply an angular velocity boundary condition to a block one would presumably be solving for the block motion.

Is there a way to apply a force on surface?
In Aria forces cannot be applied directly at a node. Instead nodal forces are computed from integrated surface tractions:

```
BC Flux for solid on surface_10 = Constant_Traction Y = 1800.0
```

What is the expression syntax for beam sections in Aria?

Syntax is similar to that of shells:

```
bar area = constant a = 0.000768
```

How can one define a temporally-ramped velocity.

The syntax should be:

```
BC LINEAR_IN_TIME Dirichlet at surface_1 velocity_x coefs = 0.1
```

If one wants to ramp up from `velocity_x = C0` to `velocity_x = C1` in time `C2` and then hold `velocity_x` at `C1` then try `BC RAMP_LINEAR_IN_TIME` instead:

```
BC RAMP_LINEAR_IN_TIME Dirichlet at surface_1 velocity_x coefs = C0 C1 C2
```

so that `velocity_x = C0 + (C1-C0) *MIN(t/C2,1.0)`. If the user manual is not handy, it is sometimes useful to extract the command syntax from the executable using:

```
sierra aria -i your.i -print-syntax | grep -i LINEAR_IN_TIME
```

This produces a somewhat cryptic definition of the expected syntax but it sometimes helps.

42.3 Errors

Cannot find a volume element on the surface error.
This problem often occurs when attempting to use a nodeset in flux boundary condition definition. Aria applies surface sideset contributions as volume contributions and often uses volume information to lookup surface information. Thus Aria is unable to look up the volume element unless it has a sideset, even though the surface nodesets are specified.

2D analysis fails due to incorrect element type used.
Typically this occurs when the grid file uses some form of 3D elements. If the grid file was created in CUBIT, then one needs to change the element type before exporting the mesh. An example of how you do that in CUBIT is:

```
block 1 element type quad
```

Additionally one must make sure to use the `dimension 2` in the CUBIT export command in order to ensure that the z coordinate is not included in the output file.

Unable to achieve convergence in anisotropic thermo-diffusion model.
This class of problems is difficult to solve. First off, the matrix is non-symmetric, which is automatically harder to solve. Second, there are different sets of physics occurring in the problem (in this case, species and temperature), and each set of physics will have a different contribution. If their contributions aren't within a couple orders of magnitude, then the problem also has great difficulty in converging. This has been viewed in thermoelectric models. In a few instances, the analyst was able to work around this difficulty by specifying a different system of units (e.g., changing from volts to millivolts). The idea here is to rescale the solution variables to be of comparable magnitude.

Nonphysical temperature overshoot in 2-material conduction model.
Typically, the temperature overshoot issue can be resolved by using `lumped_mass` instead of `mass` in the EQ specification. Use of the mass keyword invokes construction of a consistent mass matrix. While consistent mass is supported by a theoretical foundation for convergence, it oftentimes exhibits the behavior of overshooting. Most models which involve a large step change are better dealt with using a "lumped mass" matrix.

Problem with enforcing species conservation.
Usually a problem stems from inconsistent units being used in the models. Make sure the units for the reactions are consistent with everything else.

`TET10` interpolation for post-processing produces an error.
Since the `TET10` element is second-order, one needs to use `Q2` in the `EQ` specifications and for post-processing, since in both cases there is interpolation involved.

Contact resistance error: `edges share more than two element faces`.
This can occur when portions of the specified contact surfaces are physically connected. Typically these type of problems are related to the mesh when unmerge command is used. Double checking the mesh would be the first place to start.

42.4 Solver

What do `real residual check` and `time step` fails mean?

`Real Residual Check Failed` means that the solve was unable to arrive at a solution that satisfied the solve tolerance specified in the radiosity solver command block. The `Time Step Failed` message in the log file means exactly that, the Newton step has failed to converge in the number of `maximum number of nonlinear iterations` specified. This will cause the time step to be reduced and the Newton step will be retried.

Unable to achieve convergence in anisotropic thermo-diffusion model.

This class of problems is difficult to solve. First off, the matrix is non-symmetric, which is automatically harder to solve. Second, there are different sets of physics occurring in the problem (in this case, species and temperature), and each set of physics will have a different contribution. If their contributions aren't within a couple orders of magnitude, then the problem also has great difficulty in converging. This has been viewed in thermoelectric models. In a few instances, the analyst was able to work around this difficulty by specifying a different system of units (e.g., changing from volts to millivolts). The idea here is to rescale the solution variables to be of comparable magnitude.

Contact problem that is being solved is not converging.

Typically, this occurs when some blocks might not be constructing the appropriate contact constraints. By adjusting the interaction constraint tolerances might help to perform converged solves on a sub-model. The issue can be remedied by adding the following tolerances to the contact definition command block:

```
begin search options
  normal tolerance = 0.0001
  tangential tolerance = 0.00005
end search options
```

Also, the amount of contact present in a model will dictate the choice of linear solver. The recommended choice would be:

```
solution method = gmres
preconditioning method = dd-ilut
```

Aztec `numerical breakdown` error.

This specific problem occurs when the matrix near diagonal elements either go to 0 or approach a singularity. This is a common occurrence when fluids are involved.

42.5 Thermal

How to define function-based tensor thermal conductivity?

Assuming that functions are user defined functions (tabular functions). Letting the conductivity functions be `func_kx, func_ky, func_kz` respectively then the syntax for temperature dependent functions would be:

```
tensor thermal_conductivity = user_function Name_xx = func_kx Name_yy =
func_kx Name_zz = func_ky X= temperature
```

If the functions are time dependent then only difference would be: `X = time`.

Does contact conductance contribute to the Joule heating term in the energy equation?

Joule heating is applied volumetrically. Contact conductance is modeled as an interface flux condition between opposing surfaces and has no intrinsic association with a volume. Hence contact conductance cannot contribute to Joule heating.

Can Aria solve a condensation problem?

There is ongoing work in studies of evaporation, but not yet condensation.

How to evaluate the time rate change in thermal energy.

The time rate change in thermal energy should be `rho*Cp*(dT/dt)*elem_volume`. The results block should contain:

```
nodal variables  = time_derivative_at_time->TEMPERATURE as dtemp_dt
nodal variables  = pp->density as rho
nodal variables  = pp->specific_heat as cp
element variables = current_element_volume as e_vol
```

Then within the EnSight calculator you can define a variable and evaluate the following expression:

```
rho*cp *dtemp_dt*e_vol
```

How does Aria output the temperature of shell elements? Does it output the average or the outer or inner surface temperature?

For the standard shell there is only one nodal value of temperature. For the specialized implementations, linear and quadratic shells there are two nodal values of surface temperature, one associated with top and bottom of the shell.

Can one model a thermal symmetry boundary condition?

Yes, you can either specify no boundary condition on your symmetry surface, which will default to adiabatic, or specify a zero flux boundary condition, which will satisfy a symmetric solution.

How does one terminate a run based on variable rate of change (dT/dt)?

One can use the same approach for terminating a maximum rate of temperature change, dT/dt, in the same way as for using maximum temperature via Encore `MIN MAX Postprocessor` command block.

How does specify variable density and specific heat?

One can specify both variable density and specific heat with the `user_function`.

What can one model with shell elements in Aria.

Typical usage of thermal shell elements in Aria is for materials, which have no through-thickness temperature variation but still model diffusion in the plane of the shell. The shell can also account for thermal capacitance. Cases where the through thickness resistance of heat flow through the material is more important will generally be modeled with resistance contact (no shell). The in-between case is a combination of the above.

How to link reference temperature to temperature of bulk fluid element used in convective flux boundary condition.

Numerical modeling of heat transfer by enclosure radiation represents the transport between facets of the surface representation of a closed volume. Modeling this problem with enclosure radiation would require a surface representation of the body within the cavity where the closed volume is defined by the void region bounded by body and cavity. From a solution perspective, enclosure radiation heat transfer is de-coupled from the standard finite element solution. This means that one can solve the enclosure radiation problem and also include convective heat transfer from the surrounding cavity surface by computing an average body temperature and using that value as the reference temperature. Alternatively one could define a bulk element for the air cavity and setup the convective heat transfer problem to include both the cavity surface and body surface. Aria allows for superposition of heat flux contributions. In an approach with a bulk element representation of the body, the bulk element temperature would serve as the reference temperature for both convective and radiative heat transfer between the cavity surfaces and the body. If one wishes to treat the absorptive medium as being optically thin one would mesh the cavity interior and the body and use the Rosseland Mean approximation of thermal conductivity.

How to save thermal conductivity as a nodal/element variable for output.

This can be accomplished using the postprocess command. Use this command in the Region scope:

```
Postprocess Thermal_Conductivity on block_1 block_2 block_n
```

and in the Results Output, use this command:

```
Nodal Variables = pp->Thermal_Conductivity as K
```

Is it possible to use radiation enclosure and heat loss by convection using the bulk fluid element approximation?

In Aria flux contributions are cumulative. Both convection to a bulk node cavity temperature can be superposed with the enclosure radiation flux.

How to represent thermal conductivity as a polynomial function of temperature.

Since the independent variable is `Temperature`, specification would be:

```
something = Polynomial Variable=Temperature Order=2 C0=6.0789 C1=-0.0225 C2=4.0e5
```

Element Death For Laser Melting Through Plate Array.

Laser flux algorithms assume visibility of the source by the surfaces assigned in the flux definition. Hence, element death in each of the plates occurs simultaneously instead of in succession.

1. If it's a global variable you want, then add the following block in the Region scope:

   ```
   Begin Solution Options
       post process FLUX heat_conduction on surface_1 as Q_Surf
   End Solution Options
   ```

   Then you can use this as a global variable in your output blocks:

   ```
   Begin Results OUTPUT AriaOutput
       global Variables = Q_surf
   End Results OUTPUT AriaOutput
   ```

   Or in heartbeat output:

```
Begin HEARTBEAT Heart1
    variable = global Q_Surf as Q_Surf
End HEARTBEAT Heart1
```

2. If you'd like the heat flux at each node point, then add this line to the Region scope of your input deck:

   ```
   Postprocess Heat_Conduction on surface_1
   ```

   Then you can use the nodal values in your output as such:

   ```
   Begin Results OUTPUT AriaOutput
       Nodal Variables = pp->heat_conduction as Q
   End Results OUTPUT AriaOutput
   ```

All of the post-processing commands can be found in the Postprocessor Operations chapter 21 of the Aria user manual.

How to properly define contact to transfer heat.

Contact requires that the mesh be discontiguously meshed at the contact interface with normals oriented into the contact interface. In the case that `surface_1` and `surface_2` have normals oriented in the same direction, the search will fail and no contact constraints will be applied, hence no heat transfer.

Assigning a Dirichlet temperature BC to an entire block in aria.

Try the following:

```
  BC USER_FUNCTION_IN_TIME DIRICHLET ON BLOCK_3 TEMPERATURE =
UserFunctionName
```

Is there a way to apply a force on surface?

Nodal forces are computed from integrated surface tractions:

```
BC Flux for solid on surface_10 = Constant_Traction Y = 1800.0
```

Syntax for having a tensor thermal conductivity described by polynomials in both directions in Aria?

The Aria code does not currently have an option to specify tensor thermal conductivity using polynomials. Alternative are to use either:

1. User Function tensor or
2. a Calore style user sub to compute and set the tensor.

Estimating the heat flux.

For the Aria application, there a couple of ways to estimate the heat flux at selected locations. The first method is to use the `heat_flux` postprocessor on `all_blocks` while the second is to use the calculator function in EnSight.

42.6 Radiation

Viewfactor calculation doesn't converge due to bad rowsum.

Typically, this occurs when there is a problem with the mesh or the surfaces specified in the Enclosure Definition. The preferred method to debug enclosures is to add the following line to the `Enclosure Definition` Block:

```
Rowsum Database name = your_enclosure_file.e
```

Upon job execution Aria will output a file containing rowsums for each of the enclosure facets. One can then view the enclosure geometry colored by the rowsum value in order to spot the problem areas. For a good enclosure, each facet should have a viewfactor rowsum very near 1. Another option that one could try is to use the option `Use Dash Enclosures` at the Aria Region Scope. This option will "skin" all the blocks in the data set and will create all the enclosures that are contained therein.

Is it possible to use radiation enclosure and heat loss by convection using the bulk fluid element approximation?

In Aria flux contributions are cumulative. Both convection to a bulk node cavity temperature can be superposed with the enclosure radiation flux.

Is there a way to rebalance aria viewfactors from a 64 node run to be used on a 16 node run?

Currently, the view factors can be used only with the same number of processors for which they were originally computed.

Is there a way to extract the incident heat flux seen by a component within an enclosure definition?

Add these commands to the Begin Results Output block:

```
face variables = radiosity as J
face variables = irradiance as G
face variables = rad_flux as Q
```

Problem with enclosure radiation calculations.

Typically the root cause of this problem is having surface normals pointing in the wrong direction. Make sure that all sidesets have the correct surface normal.

42.7 Time Stepping

What do `real residual check` and `time step` fails mean?

`Real Residual Check Failed` means that the solve was unable to arrive at a solution that satisfied the solve tolerance specified in the radiosity solver command block. The `Time Step Failed` message in the log file means exactly that, the Newton step has failed to converge in the number of `maximum number of nonlinear iterations` specified. This will cause the time step to be reduced and the Newton step will be retried.

How does one terminate a run based on variable rate of change (dT/dt)?

One can use the same approach for terminating a maximum rate of temperature change, dT/dt, in the same way as for using maximum temperature via Encore `MIN MAX Postprocessor` command block.

Time step size not increasing.

Even after setting the maximum time step, the time step might be limited or constrained by the output increment in the `Begin Results Output` block.

Aria fails to exit when timestep refined to zero.

Simple workaround is after the `Initial Time Step Size = 5` command, add the syntax `Minimum Time Step Size = 5`. This will prevent Aria from trying to half the time-step and will force it to exit.

Early-time results of steady-state run are nonphysical.

When one runs a linear to mildly non-linear model in a pseudo-transient manner, the intermediate results obtained will generally appear unreasonable and even nonphysical. Although it may appear that physical time steps are being taken, the only result that can be relied upon is the steady-state solution, as long as the solution has converged.

How can I transition a tabular material property from steady-state to transient?

The following input is sufficient for changing a material property between steady-state and transient analysis in the same simulation. Note that the steady-state material property goes from 0.0 to 1.0, and the transient portion comprises 1.0000001 and onward.

```
begin definition for function mat1_time
    type is piecewise linear
    abscissa = x_label
    ordinate = y_label
    begin values
        0.000000    0
        1.000000    0
        1.0000001 129
        1e9       129
    end   values
end
```

42.8 Contact

What are the proper units for electrical contact conductance coefficient?

The units are equivalent to a heat transfer coefficient which would have the units of 1/(resistance*area), which is equivalent to (current/(volts*area)).

Can I define contact conductance as an analytic function?

No, this capability is not currently available.

Can one simulate a sliding block in Aria?

Yes, one can do this but the contact physics will be based strictly upon contact between the the two surfaces, without sliding friction effects.

How to model multiple thermal and electrical contact conductances.

Typically, attempts to use two contact definitions in the Aria Region fail due to conflicts between the conductance coefficient used in each of the contact definitions. This problem can be circumvented by using multiple Aria Regions or by defining multiple sidesets at the contact interface.

Contact problem that is being solved is not converging.

Typically, this occurs when some blocks might not be constructing the appropriate contact constraints. By adjusting the interaction constraint tolerances might help to perform converged solves on a sub-model. The issue can be remedied by adding the following tolerances to the contact definition command block:

```
begin search options
  normal tolerance = 0.0001
  tangential tolerance = 0.00005
end search options
```

These tolerances reflect physical distance which should represent the the geometry of the modeled discretization. Also, the amount of contact present in a model will dictate the choice of linear solver. For generalized contact, `tied_temperature` or `gap_conductance` the recommended solver is:

```
solution method = gmres
preconditioning method = dd-ilut
```

What are the proper units for electrical contact conductance coefficient?

The units are equivalent to a heat transfer coefficient which would have the units of 1/(resistance*area), which is equivalent to (current/(volts*area)).

42.9 Species

Does Aria support modeling a reaction that is dependent on temperature, pressure, and species concentration?

Yes, it is possible to implement a user plugin for quantities that are dependent on each of those.

Problem with enforcing species conservation.

Usually a problem stems from inconsistent units being used in the models. Make sure the units for the reactions are consistent with everything else.

Species diffusion subroutines cause non-physical results and residuals are high.

Try using a direct solver. At times, GMRES with Jacobi or DD-ILUT for preconditioning methods may not converge.

42.10 Porous Media

How can one output the non-wetting phase pressure?

Since the non-wetting phase pressure is needed in the wetting phase solution, it is available for output not as a solution variable but as a post-processed variable. In general, the pressure can be output provided one is able to specify its material phase. In this case the syntax would be:

```
postprocess non_wetting_phase_pressure on all_blocks
```

and the variable could be output to a results file as:

```
nodal variables = pp->non_wetting_phase_pressure as P_NW
```

Can capillary pressure be added as postprocess variable?

Yes, post-processing of capillary pressure is supported, which can later be added to an output.

No expression for intrinsic permeability vs. time table.

Intrinsic permeability is handled a bit differently than other porous media properties since the tensor character (ratio of coefficients) is normally preserved. Instead intrinsic permeability is modified by introducing a scaling factor:

```
Intrinsic Permeability = Constant XX=1.0 YY=1.0 ZZ=1.0
Intrinsic Permeability scaling = user_function name = Time_Permeability_Function x = time
```

What is the difference between the options `Porosity = Mesh_deforming` and `Porosity = Solid_deforming`?

`Mesh_Deforming` and `Solid_Deforming` are identical, except `Mesh_Deforming` works on the `mesh` equation and `Solid_Deforming` works on the `solid` equation.

What do all the different forms of density mean?

For porous media problems, a typical input file may contain specifications for many types of density. The common instances are `DENSITY`, `SOLID_DENSITY`, `BULK_MASS_DENSITY`, and `BULK_DENSITY`.

The units for `DENSITY` are different for solid and non-solid phases. For the solid phase, `DENSITY` is mass per element volume, while for non-solid phases it is mass per phase volume.

The `SOLID_DENSITY` is an intrinsic material property, defined as the mass of a substance divided by the volume occupied by that substance. In Aria it is only a valid property for the solid phase, and is typically used to define a variable porosity in combination with the solid mass fractions.

The `BULK_MASS_DENSITY` is in units of mass per element volume for all phases. This is used in the mass term for mass balance equations, and can be selected from a number of different models. Because of the unit difference in `DENSITY` between solid and non-solid phases not all models are applicable in all phases. The `POROUS_DENSITY`, `MASS_FRACTION_POROUS_DENSITY`, and `MULTIPHASE_POROUS_DENSITY` models are only valid in non-solid phases while the `MASS_FRACTION_DENSITY` and `DENSITY` models are only valid in the solid phase. An error will be generated if you attempt to use the wrong model for a given phase.

The `BULK_DENSITY` has the same units as `BULK_MASS_DENSITY`. It is primarily used as a convenient way of specifying an initial condition that implicitly includes porosity. It can be used for the solid phase to initialize `DENSITY` using the `MASS_AVERAGE` model.

42.11 Electrostatics

What are the proper units for electrical contact conductance coefficient?

The units are equivalent to a heat transfer coefficient which would have the units of 1/(resistance*area), which is equivalent to (current/(volts*area)).

Is Aria capable of calculating and outputting an electromotive force resulting from a body in an electric field?

Currently, there is no native postprocessor in the Aria code to compute resultant electromotive force.

42.12 Grid Refinement

Can Aria do focused mesh refinement?

Yes, the manner in which the refinement occurs is controlled by the `marker_field`. Most of the adaptivity support is finite element based, single level of refinement and is driven by Encore. The necessary commands include:

```
Begin Sierra Aria
  Begin Uniform Marker uniform_mark
    Store in marker_field
  End   Uniform Marker uniform_mark

  Begin Procedure AriaProcedure
    Begin Solution Control Description
      Use System Main
      Begin System Main
        MarkAdapt AriaRegion using uniform_mark
        Begin Transient TransientBlock
          Advance AriaRegion
          Begin Parameters for Transient TransientBlock
          ...
```

When using uniform mesh refinement, does the new mesh get saved somewhere?

The new mesh is on the output from the uniform mesh refinement run. The disadvantage with that is the file size with multiple time planes in it. A more economical option is to use an additional Results Output block that saves zero time plane with the refined mesh. The syntax would be something like that in the example where the `At Step 0 Interval = 10e10` being ridiculously high should cause only one time plane to appear in the output file. One may also use the Sierra executable `stk_adapt_exe` for uniform mesh refinement - documentation for this application may be found on Compsim.

Problem encountered with using 8-node quad.

In order to support mesh refinement in the same way for all elements, Aria does not support the 8-node quad. It only supports 9-node element.

How does one coarsen a mesh?

The input modifications consist of two new lines:

```
MarkAdapt AriaRegion Using total\_mark When "recover_ind > 0.5"
MarkAdapt AriaRegion Using total\_mark2 When "recover_ind < 0.25"
```

This syntax turns on refinement when the error indicator exceeds the value 0.5 and switches to coarsening when the error is less than 0.25. When the error is between these values, the mesh is left unchanged.

42.13 Post Processing

How can one output the non-wetting phase pressure?

Since the non-wetting phase pressure is needed in the wetting phase solution, it is available for output not as a solution variable but as a post-processed variable. In general, the pressure can be output provided one is able to specify its material phase. In this case the syntax would be:

```
postprocess non_wetting_phase_pressure on all_blocks
```

and the variable could be output to a results file as:

```
nodal variables = pp->non_wetting_phase_pressure as P_NW
```

How to output the min and max temp on specified blocks.

This can be accomplished using Encore:

```
$-------------------------------------------------
$   Specify Encore output to text file
$-------------------------------------------------
Begin Postprocessor Output Control Encore_Out
  Output Time
  Write To File TempMinMax.txt
  Floating Point Precision Is 3
  Floating Point Format Is Fixed
End   Postprocessor Output Control Encore_Out

$-------------------------------------------------
$   Create Encore field function
$-------------------------------------------------
Begin Field Function spos
  Use Nodal Field Solution->Temperature
End   Field Function spos

$-------------------------------------------------
$   Determine maximum value of field function
$-------------------------------------------------
Begin Min Max Postprocessor TMinMax_1
  Use Function spos
  Compute Min Max
  Volumes block_1
End   Min Max Postprocessor TMinMax_1

Begin Procedure AriaProcedure
  Begin Aria Region AriaRegion
    Evaluate Postprocessor TMinMax_1
...
```

How to output mass inventory.

Include the following commands at the Aria Region scope:

```
Begin solution options
```

```
    post process mass
    post process volume
End
```

Model hangs when trying to output elem_temperature.

The proper syntax for postprocessing Temperature as an element variable is:

```
Postprocess TEMPERATURE on all_blocks using P0
```

This will interpolate the nodal temperature as a single element value, a zeroth-order polynomial (`P0`).

How to evaluate the time rate change in thermal energy.

The time rate change in thermal energy should be `rho*Cp*(dT/dt)*elem_volume`. The results block should contain:

```
nodal variables  = time_derivative_at_time->TEMPERATURE as dtemp_dt
nodal variables  = pp->density as rho
nodal variables  = pp->specific_heat as cp
element variables = current_element_volume as e_vol
```

Then within the EnSight calculator you can define a variable and evaluate the following expression:

```
rho*cp *dtemp_dt*e_vol
```

How does Aria output the temperature of shell elements? Does it output the average or the outer or inner surface temperature?

For the standard shell there is only one nodal value of temperature. For the specialized implementations, linear and quadratic shells there are two nodal values of surface temperature, one associated with top and bottom of the shell.

How to save thermal conductivity as a nodal/element variable for output.

This can be accomplished using the postprocess command. Use this command in the Region scope:

```
Postprocess Thermal_Conductivity on block_1 block_2 block_n
```

and in the Results Output, use this command:

```
Nodal Variables = pp->Thermal_Conductivity as K
```

Can capillary pressure be added as postprocess variable?

Yes, post-processing of capillary pressure is supported, which can later be added to an output.

How to visualize the results continuously regardless of the time step selected by ARIA?

To visualize the results of an adaptively time-stepped simulation in an equal-time-step fashion, one can use a ParaView filter, "Temporal Interpolator". This filter allows one to view a time-dependent data set where the data's field data between adjacent time steps is linearly interpolated.

How to output an ASCII history file for data at a selected location over time.

One way of getting the history output that one specifies is to use Encore to do the interpolation. Graphics packages such as EnSight and ParaView can then be used to generate time history of an output parameter at selected locations.

`TET10` interpolation for post-processing produces an error.

Since the `TET10` element is second-order, one needs to use `Q2` in the `EQ` specifications and for post-processing, since in both cases there is interpolation involved.

How to output element block volumes for steady-state simulation.

In order to output volume and mass in Aria one uses the following command block in the Aria Region:

```
Begin Solution Options
    post process volume
End   Solution Options
```

Log file has unwanted postprocessor history at every time step.

Remember to comment out the following:

```
begin postprocessor output control test
    output to log file
end   postprocessor output control test
```

Is there a way to apply a force on surface?

Nodal forces are computed from integrated surface tractions:

```
BC Flux for solid on surface_10 = Constant_Traction Y = 1800.0
```

Using Tecplot to plotting/visualize Aria results.

One can use exotec2 - the syntax is:

```
exotec2 exo_in tec_out
```

where `exo_in` is the exodus file and `tec_out` is the Tecplot translation.

Is there a way to extract the incident heat flux seen by a component within an enclosure definition?

Add these commands to the Begin Results Output block:

```
face variables = radiosity as J
face variables = irradiance as G
face variables = rad_flux as Q
```

Estimating the heat flux.

For the Aria application, there a couple of ways to estimate the heat flux at selected locations. The first method is to use the `heat_flux` postprocessor on `all_blocks` while the second is to use the calculator function in EnSight.

Chapter 43

Developer Documentation

When working within the code base developers may wish to setup their environment to reference the development toolset.

43.1 Setting The Environment-Developers at Sandia Labs

The environment for using Aria 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-devel
```

43.1.1 Setting up for the `csh` and `tcsh` Shells

Add SNTools to your path. In either your `/.cshrc` or `/.tcshrc` file add the line

```
set path=(/home/sntools/devel/linux/install/sntools/engine $path)
```

43.1.2 Setting up for the `bash` Shell

Add SNTools to your path. In either your `/.bashrc` or `/.profile` file add the line

```
export PATH=/home/sntools/devel/linux/sntools/engine:$PATH
```

43.2 An Introduction to Aria's Expression System

The following is the conference paper from the Coupled Problems 2005 conference, See [46].

This section provides a brief overview of a core element of Aria's design called the Expression system. Understanding the Expression system is not essential for using Aria but it is useful in understanding how some of Aria's features behave. It is also the simplest point of entry for developing and extending Aria with new material properties and boundary conditions. In short, the Expression system is the abstraction of low-to mid-level numerical calculations so as to make the code highly general, reusable, extensible and scalable.

In order to provide a low entry barrier for user extensions we have chosen a design where the essential numerical entities of boundary conditions, material properties, etc., are all implemented the same way.

Specifically, these are implemented as C++ classes called Expressions. In addition to supplying the functional evaluation of the numerical quantity, Expression implementers provide information regarding what the Expression provides, what other Expressions it may depend on, the tensorial order of the provided quantity, etc. A higher-level Expression Manager is then able to determine the minimal and sufficient set of Expressions required to perform the simulation and to ensure that they are evaluated in the proper order. In addition to providing a simple extension technique, this makes the core development of Aria very clean and abstracts nonessential details from many of the basic algorithms, while still employing the most efficient numerical kernels and data management techniques at the lowest levels.

To motivate our approach, we start with a simple example. In the FEM solution of the Navier-Stokes equations the following integral, among others, is computed.

$$\int_{\mathrm{V}} \rho \boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{v} \phi^i \, \mathrm{dV} = \sum_e \int_{\mathrm{V}_e} \rho \boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{v} \phi^i |j| \, \mathrm{dV}_e \tag{43.1}$$

Here ρ is the density, $\boldsymbol{v}$ is the fluid velocity and ϕ^i is weight function associated with local node i. V is the domain of integration in the physical space and V_e is that in the space of the element with $|j|$ being the Jacobian of transformation from V_e to V. Thus, the assembly kernel responsible for this calculation needs access to all of these quantities. We will show that the kernel, however, does not need to know the particulars of, say, the density model or of the weight function, especially regarding which other quantities they may depend on. Moreover, there are likely to be other assembly kernels that may also require these same quantities, such as the advection term of the energy transport equation,

$$\int_{\mathrm{V}} \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T \phi^i \, \mathrm{dV} = \sum_e \int_{\mathrm{V}_e} \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T \phi^i |j| \, \mathrm{dV}_e \tag{43.2}$$

where C_p is the specific heat and T is the temperature. Efficient evaluation of (43.2) should make use of $\rho, |j|, \boldsymbol{v}$ and any other quantities that might have previously been computed in the assembly of (43.1). Keeping track of which quantities are needed and which quantities have already been evaluated in a previous part of the assembly, regardless of the equation system that is being solved, is the purpose of Aria's Expression Manager.

In Aria, all of the quantities presented above, ρ, $\boldsymbol{\nabla} \boldsymbol{v}$, $|j|$, ϕ^i, etc., are implemented as C++ classes derived from the base class Expression. Each Expression has these basic responsibilities:

- **Identification.** Each Expression identifies itself with a generic description, such as density and, when applicable, a specific model description, such as cubic_in_temperature.
- **Prerequisites.** Each Expression (and Expression user) provides a list of other Expressions (using the generic identifiers) that are required for its own calculations. To continue the example, the Expression which implements the cubic_in_temperature model would have temperature as a prerequisite. Arbitrarily complex terms may be built of successively simpler components which ultimately result in a dependency tree with the simplest ingredients being the leaves of the graph.
- **Tensor Properties.** Each Expression states its tensor order (scalar, vector, second order tensor, etc.) and dimension. With this information, Expression users can dynamically determine how to index the values provided by the expression. For example, some thermal conductivities may be scalars while others may be second order tensors as is the case with anisotropic materials.
- **Evaluation.** Each Expression implements a virtual function for computing its values. Depending upon the nonlinear solution strategy, additional methods may be required for computing, for example, Newton sensitivities.

When an Expression is created, it makes itself known to an Expression Manager. The Expression Manager is responsible for establishing the minimal but sufficient list of Expressions required for the calculation. This is done utilizing the lists of prerequisites provided by Expression users and other Expressions. The relationships

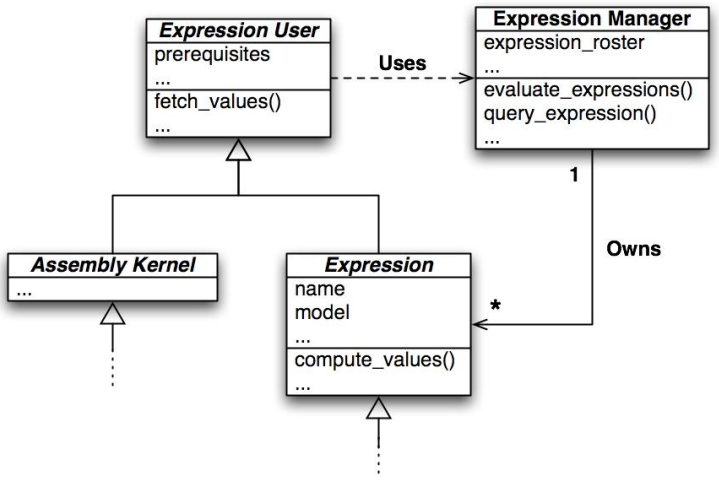


Figure 43.1. Schematic UML class diagram for the Expression subsystem.

among Expressions, Expression users, and the Expression Manager is illustrated schematically in figure 43.1 with the use of UML. This figure illustrates two salient features of our Expression subsystem: Expressions are themselves Expression users and Expression users, such as assembly kernels, are the original source for all prerequisites.

The richness of the information provided in these lists of prerequisites quickly becomes apparent. For example, the Expression Manager is able to further utilize the prerequisite information to order the list of Expressions for evaluation. This is done so that when an Expression is evaluated, all of its prerequisite Expressions have already been evaluated. Regardless of how complicated the multiphysics problem happens to be, there are no duplicate evaluations and no overhead costs that would be incurred with lazy-evaluation. Another powerful use for the prerequisite information is in dynamically determining and computing sensitivities for Newton's method. Since primitive variables, or degrees of freedom, are also represented as Expressions, the prerequisites for any given Expression are recursively checked to see if any of them represents a primitive variable. In combination with runtime-queried tensor properties (see above), an Expression developer can implement and evaluate the Expression's sensitivities without ever having direct knowledge of the degrees of freedom in the system. Lastly, Aria utilizes this information for additional purposes such as sparse matrix allocation and automated memory management.

43.3 Nonlinear Coupling Strategies in Aria

One of the difficulties with writing broadly applicable computational mechanics software is that developers can't take advantage of specific knowledge of the application domain in order to optimize the algorithm. Thus, in providing generality one sometimes sacrifices efficiency. One place this is evident in multiphysics modeling is in the choice of coupling strategies. While it is well understood that a fully coupled system solved with Newton's method utilizing analytic sensitivities is formally the most robust and correct approach to solving multiphysics applications it is also computationally expensive and complex to implement. Furthermore, while Newton's method has the fastest rate of asymptotic convergence it's domain of convergence is often empirically observed to be smaller than other methods. Lastly, in some applications, certain subsets of the physics may be only weakly coupled so that a loosely coupled approach may be more computationally efficient. To address these concerns while remaining general and flexible Aria offers a number of options for

nonlinear solution strategies and physics coupling.

In defining a problem in Aria, users configure one or more Regions. Each Region consists of one or more PDEs to be solved on some or all of the input mesh. All of the PDEs in each Region are solved in a tightly coupled (i.e., single matrix) manner using one of several nonlinear solution strategies available. Users may then 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 particulars 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, an Aria 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. First, the resulting linear system is considerably smaller and contains far fewer off-diagonal contributions which can significantly increase the performance of linear solvers. Also, a resulting linear system may have a more attractive form, such as symmetric positive-definite, that 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 Sierra applications.

43.4 Developer Recipes

43.4.1 Adding a New Flux Boundary Condition

1. Add a new name for your model
   (a) Add a new `Identifier` entry to `Aria_Model_Names.h`
   (b) Add the string in `Aria_Model_Names.C`
2. Add your new Expression class.
   (a) Copy-paste-rename a similar Expression definition. See, e.g., `Aria_Material_Models.h`.
   (b) Copy-paste-rename a similar Expression implementation. See, e.g., `Aria_Material_Models.C`.
3. Add an entry to the `Expression_Factory::create_model_expression()`.
   (a) Copy-paste-rename a similar entry in `Aria_Expression_Factory.C`.

43.4.2 Adding a New Material Model

1. Add a new name for your model
   (a) Add a new `Identifier` entry to `Aria_Model_Names.h`
   (b) Add the string in `Aria_Model_Names.C`
2. Add your new Expression class.
   (a) Copy-paste-rename a similar Expression definition. See, e.g., `Aria_Material_Models.h`.
   (b) Copy-paste-rename a similar Expression implementation. See, e.g., `Aria_Material_Models.C`.
3. Add an entry to the `Expression_Factory::create_model_expression()`.
   (a) Copy-paste-rename a similar entry in `Aria_Expression_Factory.C`.

43.5 Expression Reference and API

This section provides a high-level view of how Expressions are run by Aria, from creation to computation. This section also provides a brief description of the Expression methods that a user either must or can optionally supply.

43.5.1 Execution Sequence

1. Initialization Phase (Input Parsing Time)

   (a) `Expression::Expression()` (See 43.5.2) : Constructor. Called at creation time. Defines identity, dependents and parameters. Required.

   (b) `Expression_Manager::preprocess_expressions()` : Called after all expressions have been created and all dependencies have been registered/requested. Unused expressions are destroyed. Dependency based ordering established.

   (c) `Expression::set_nonlinear_method()` : Sets the nonlinear method that will be used.

   (d) `Expression::create_dynamic_storage()` : Initializes storage for values and Newton sensitivities but does not actually allocate memory.

   (e) `Expression::postregistration_setup()` (See 43.5.3) : Called once, after the Expression_Manager has determined that the Expression will indeed be used but before simulation begins. Called in dependency order. Can get references to dependent's storage (values, Newton sensitivities, etc.).

   (f) `Kernel::postregistration_setup()` : Assembly kernels can interrogate the Expressions available to them and get references to values and Newton sensitivities.

2. Simulation Phase

   (a) `Expression::prepare_to_recompute()` (See 43.5.4): Called once for each workset, immediately prior to compute_values(). Can be used to resize or reinitialize storage (usually done by base class), inquire about simulation time, etc.

   (b) `Expression::compute_values()` (See 43.5.5) : Computes the values of the Expression. Required to compile.

   (c) `Expression::compute_sensitivities()` (See 43.5.6) : Computes the Newton sensitivities of the Expression. Required to compile.

3. Shutdown Phase

   (a) `Expression:: Expression()` (See 43.5.7) : Destructor. Can be used to free memory allocations (note: Real_MDArray storage is automatically de-allocated). Required but normally empty.

43.5.2 Constructor

43.5.3 `postregistration_setup()`

43.5.4 `prepare_to_recompute()`

43.5.5 `compute_values()`

43.5.6 `compute_sensitivities()`

43.5.7 Destructor

43.6 Newton Sensitivity Checking for Expressions

When you run Aria you can ask it to look for sensitivity errors in every Expression if you're using Newton's method. If activated, Aria will compare the analytic sensitivities provided by each expression with numerically computed values. This feature is enabled by adding the command line option `-arialog sens_check` to Aria. If you're using the `sierra` tool to run Aria, then add `-O "-arialog sens_check"` (including quotes) to the `sierra` command line. The same holds if you're running the Arpeggio application.

You can also run the entire test suite with Jacobian checking enabled by adding `-u aria_args = "-arialog sens_check"` to the `runtest` command line.

43.7 Profiling

It's a good idea to profile the code before spending time trying to optimizing and performance tuning. Fortunately, profiling is pretty easy to do. On Linux, one approach is to use the `gprof` tool. To profile code with gprof, you'll need to compile with the `-pg` option. The easiest way to do this is to set environment variables `USER_CFLAGS` and `USER_LDFLAGS` to `-pg`. In the bash/ksh/zsh shell, use, e.g., `export USER_CFLAGS=-pg` and in tcsh/csh use `setenv USER_CFLAGS -pg`. You'll only get detailed profiling information for code compiled and linked with this flag so you may want to recompile some of the framework too. You can profile debug or optimized code.

With the instrumented executable, run Aria as you normally would. This will generate a file called `gmon.out`. Lastly, run `gprof` to analyze the data: `gprof` *`executable`* `> gprof.txt` where *`executable`* is the executable (probably including an absolute or relative path) used in the run. The output stored in `gprof.txt` will give you some real data telling where the real bottle necks are.

43.8 Purify: Memory Analysis and Debugging

1. Build Aria on Linux. This works for both optimized and debug modes but use debug if you want to see line numbers in Purify.
2. Copy the complete link line into a file.
3. Add `-show` option to `mpiCC`.
4. Remove the `-static` option from the link line arguments.

5. Adjust any path to `Apps_aria.o` if necessary.
6. Put `which mpiCC` on the previous line to verify you're getting `mpiCC` from `/usr/local/mpi/sierra/mpich/1.2.5.2/gcc-3.2.2/bin/`.
7. You might also verify `gcc` is version 3.2.2 (`gcc -v`).
8. Source the script.
9. Copy the link line (start with `g++ ...`) and paste it into a file.
10. Preface `g++` with `purify`, e.g., `purify -always-use-cache-dir -cache-dir=‘pwd‘/ g++ ...`
11. Source this script and it will instrument your executable. It takes a while...
12. Optionally verify that the executable is `purify`ed by running `ldd aria.x`
13. Copy executable to aria/bin directory, removing the `.tmp` extension.
14. Run it standalone (via `sierra`, `runtest` or by hand) or with TotalView.

43.9 Building Against Other Projects

Sometimes it can be very useful to build against changes that have been made in a different project. For example, if you are waiting for a project's changes to be checked in but would like to start developing against those changes in another project. To do this with the SNTools `build` tool you can use the `-a` option to build. The argument to this option is the XML file for each *product* you want to include in your build. So, if you want to use the version of Krino found in a project located in `/path/to/project/` then you would add `-a /path/to/project/krino/krino_sn.xml` to your normal `build` command line.

43.10 Interfacing with MATLAB

43.10.1 Reading Aria Matrices Into MATLAB

This is sort of an "old" way of getting matrices into MATLAB but it still works just fine.

From Matt Hopkins:

You can read a `.mtx` matrix into MATLAB via the following. Let `my_matrix.mtx` be the matrix sitting in the debug output path you're interested in.

```
>> A_ascii = load('my_matrix.mtx');
>> A = spconvert(A_ascii(2:end,:));
```

And now `A` is in MATLAB's sparse matrix format. It even ignores the zeros in the `.mtx` file (does not allocate space for them).

From there you can do some pretty nifty things. `spy(A)` does sort of what matvis does (graphical view of matrix nonzeros). `condest(A)` estimates the 1-norm condition number of `A` (might take a while for large `A`). Etc.

43.10.2 Interfacing Aria from within MATLAB

Russell Hooper is developing a pretty sophisticated MATLAB interface that allows you to control Aria and probe nonlinear and linear solver data structures directly from within MATLAB. This is rapidly evolving so no details are given here... except for how to build and run Aria/MATLAB.

This recipe is taken from an email exchange between Matt Hopkins and Russell Hooper in early April 2006. It most likely only works on Linux.

```
% setenv LD_LIBRARY_PATH  /usr/local/matlab/7.2/bin/glnx86:\${LD_LIBRARY_PATH}

% build aria -j8 -a /home/rhoope/Trilinos_6.0  \
   CXXFLAGS="-I/usr/local/matlab/7.2/extern/include/" \
   -a  /home/rhoope/projects/noxMatlabStable/equationsolver \
   LDFLAGS="-L/usr/local/matlab/7.2/bin/glnx86  -leng"
```

If you want to build a debug version, add "-g" to both the `CXXFLAGS` and `LDFLAGS`.

To execute aria, first invoke `sierra aria -i input.i -script-only`. Then copy the file `sierra.sh` into another, ie `opt.sh`. Finally, remove the "-o logfilename" argument from `opt.sh` so that your run is truly interactive.

43.11 Error Handling

Traditional techniques for error handling of numerous and varied. Sometimes functions will return special values indicating success or failure of the call. Sometimes global variables or flags are set to indicate an error has occurred. Sometimes abort() is simply called (bad!).

C++ and the SIERRA Framework offer facilities to support error handling. The two primary error handling techniques are *exception handling* and *assertions*. Both of these can be combined to implement *Design by Contract* though this is not formally done in Aria.

Whether one should use an exception or an assertion is sometimes a subtle question. In general, assertions should be used for cases where the error is not likely to occur due merely to user input. However, assertions can also be useful for computationally expensive tests which would adversely affect production calculations.

43.11.1 Exception Handling

C++ offers *exception handling* and the SIERRA Framework utility library provides a parallel-safe layer for exception handling. Because exceptions in C++ are ordinary classes, applications may choose to specialize exceptions for handling and detecting specific failure modes. When SIERRA exceptions are used in conjunction with the SIERRA diagnostic tracing facility, it is possibly to get a stack trace (or sometimes a partial one) that illustrates the code path the where the exception was thrown.

In most cases, Aria developers do not need to worry about catching exceptions (they're normally fatal errors). The top-level SIERRA Framework routine `run_sierra()` handles the responsibility of catching any uncaught exceptions, converting them into parallel exceptions if necessary, and propagating the exception to any other processors.

If the reader is not already familiar with exception handling in C++, a good place to start is with [47]. In it's simplest form, SIERRA exceptions can be used as follows.

```
#include <util/Exception.h>
// ...
if(something_bad_happened)
  {
    sierra::Exception x("Nice descriptive error message.");
    throw x;
  }
```

The `sierra::Exception` class inherits from `std::string` and so it supports all of `std::string`'s operations, most notably the + (concatenation) operator.

```
#include <util/Exception.h>
// ...
if(something_bad_happened)
  {
    sierra::Exception x("Nice descriptive error message.");
    x += "  More info here.";
    throw x;
  }
```

Additionally, the `sierra::Exception` class supports the put-to operator, `«`, for stream-like handling.

```
#include <util/Exception.h>
// ...
if(something_bad_happened)
  {
    sierra::Exception x;
    x << "Nice descriptive error message."
      << "  More info here.";
    throw x;
  }
```

However, in order to keep the `sierra::Exception` class light weight, it does not inherit from `std::ostringstream`. Instead, `operator«` is only defined for the most common objects, such as strings, integers, floats/doubles, etc. The consequence of this is that if you define a class with `operator«` for `ostream`, that operator will not work with `sierra::Exception` objects. In these cases, it is sometimes useful to use a `std::ostringstream` to construct the error message and then pass the resulting `std::string` to the exception.

```
#include <util/Exception.h>
#include <sstream>
// ...
if(something_bad_happened)
  {
    std::ostringstream os;
    os << "Nice descriptive error message."
       << "  Object foo: ";
       << some_object
       << " is unusable";
    sierra::Exception x;
```

```
    x << os.str();
    throw x;
  }
```

Finally, the construction and throwing of the exception can occur on the same line for brevity. In this case, all we need is a temporary object which is unnamed.

```
#include <util/Exception.h>
// ...
if(something_bad_happened)
  {
    throw sierra::Exception() << "Nice descriptive error message."
                              << " Integer out of bounds: j = "
                              << j;
  }
```

43.11.2 Assertions

Assertions are a common way to ensure that certain conditions that are assumed by the code to be true are indeed true. For example, a function or piece of code may require that a pointer be non-NULL, an integer be greater than zero, etc. Using the C++ `assert()` macro is a good way to test such conditions that *should* hold under normal circumstances. By using `assert()`, the enclosed test will only be performed if the code is compiled in debug mode; in optimized mode, the macros expand to nothing. Thus, developers can use asserts extensively to test conditions assumed by their code without affecting production-mode performance. Assertions also provide an excellent way to document *and enforce* the assumptions that the developer made in writing the code.

The SIERRA Framework provides an alternate implementation of the `assert()` macro that utilizes the parallel-safe exception handling described above. The macro, `ThrowAssert()`, is defined in the same header file as the exception class.

```
#include <util/Exception.h>
// ...
void
some_function(const MyClass * myclass_ptr)
{
  ThrowAssert(0 != myclass_ptr);
  // It should be OK to use the pointer.
  // ...
```

Because assertions expand to empty code in optimized mode, developers must be careful to never put variable assignments or other state-altering code inside an assert.

```
#include <util/Exception.h>
// ...
// This is a bug:
ThrowAssert(j = i + 2 > 4);
```

A common trick to getting more useful messages from a failed assertion is to add a help string like this:

```
#include <util/Exception.h>
```

```
// ...
void
some_function(const Int & num)
{
  ThrowAssert(num > 10 &&
              "This function only works propertly when num is larger than 10");
  // It should be OK to use the pointer.
  // ...
```

Lastly, another macro, `ThrowRequire()` is available. This macro is identical to the `ThrowAssert()` macro except this it is always enabled, even in optimized builds. This macro is currently not used in Aria but use it if you want. The concise and readable form of these macros can help keep code short while still being readable and expressive.

43.12 Outputting User Information (Logging)

The SIERRA Framework utility library supplies a flexible logging facility. The problem with normal `printf` and `cout` output functions is that they get really annoying in parallel. Further, these messages simply roll off the top of the user's screen unless they are careful to redirect the standard output and error streams to a file. Lastly, it's difficult, and may involve recompiling the application, to tailor these messages depending on the kind of information a user is interested in.

43.12.1 The DiagWriter Logging Facility

SIERRA's logging facility, `DiagWriter`, is designed to address these three issues. The `DiagWriter` class, in the simplest sense, is a `std::ostream` class that supports the put-to (`«`) operator for writing messages.

```
#include <Aria_DiagWriter.h>
//...
namespace Aria
{
  arialog << "This is a message the user will always see." << std::endl;
}
```

Note the use of `std::endl` instead of `std::endl` (the standard and portable newline). This is an unfortunate but important point; using `std::endl` will result in ugly compiler errors.

The most interesting feature of the `DiagWriter` class is that the output can be tagged with a bit field called a *message mask*. These masks are defined in Aria_DiagWriter_fwd.h as enums with meaningful names, e.g., `LOG_EXPRESSION` and `LOG_NONLINEAR`. To mask your output message, you can use the `m()` method on the `DiagWriter` object.

```
#include <Aria_DiagWriter.h>
//...
namespace Aria
{
  arialog.m(LOG_EXPRESSION)
     << "This message is supposedly related to Expressions."
     << std::endl;
  arialog.m(LOG_EXPRESSION | LOG_NONLINEAR)
```

```
      << "This message is supposedly related to Expressions "
      << "and/or the nonlinear solver"
      << std::endl;
}
```

Messages that contain a mask are only written if that is enabled via the *print mask* and messages that don't have a specified mask are always written. The user can define a print mask using the "-arialog *string*" command line argument where the provided string maps to one (or more) of the bit masks. The file Aria_DiagWriter.C contains the mapping between the string names and the bit mask values. For example, "expression" is assigned to `LOG_EXPRESSION`. When the user supplies the command line argument "-arialog expression" then the messages written are those masked with `LOG_EXPRESSION` and those with no bit masks.

The `DiagWriter` class contains many more features that are beyond the scope of this section. Explore the header files and the code for more examples.

43.12.2 The Tracing Facility

Tracing is a common and extremely useful debugging utility. With tracing enabled, the code prints the name of each function as it enters and exits the function. C++ doesn't provide any standard way to accomplish this so the `Trace` class is provided by the SIERRA Framework utility library.

The `Trace` class works by creating an instance (object) as the first line of a. The object constructor takes as an argument a string (char *) containing the name of the function. When tracing is enabled the constructor prints this name. When the application leaves this function, the local `Trace` object is automatically destroyed as it goes out of scope. When tracing is enabled, the automatically-called destructor re-prints the function name. The output of nested functions is nested so the output provides a hierarchical view of the flow of control. The most basic usage looks like this:

```
#include <Aria_DiagWriter.h>
//...
namespace Aria
{
  void my_func(const Int & i)
    {
      Trace diag_trace("Aria::my_func(const Int & i)");
      //... normal code follows
    }
}
```

The `Trace` class utilizes the `DiagWriter` class for writing and for determining if tracing is enabled. So, tracing may be enabled using "-arialog trace" on the command line and `Aria::Trace` objects mask all output with `LOG_TRACE`.

Though it's not discussed here, tracing can be turned on and off during the normal execution so that tracing information can be gathered only when it's desired. Read through the header files and the code for examples of doing that. One example of where this is used is in the exception handling. When an exception is thrown, the tracing bit mask is automatically turned on. As the function stack unwinds during the throw, any functions instrumented with `Trace` objects will get destructed and hence they will print their owning function name. The result is a stack trace, or traceback, which can be extremely useful for debugging.

Adding all of the Trace objects to a code can be a daunting task, especially if you're really anal and want the function arguments and namespace to be a part of the function's printed name (which can be important with polymorphic functions). Aria uses the traceString tool (http://tracestring.sourceforge.net/)

to automatically instrument the code with the `Trace` objects. The traceString tool will enclose the `Trace` objects in a pair of special strings so that it can safely update or remove the inserted code later on. Typically, it looks like this:

```
#include <Aria_DiagWriter.h>
//...
namespace Aria
{
  void my_func(const Int & i)
    {
      /* %TRACE[ON]% */ Trace diag_trace("Aria::my_func(const Int & i)"); /* %TRACE% */
      //... normal code follows
    }
}
```

To learn more about traceString, talk to Pat...

43.13 Catalogue of Assembly Kernel Expressions

Aria supplies several generic expressions that can be used for top-level assembly kernels for equation terms. When adding new equations or terms to existing equations, one of the follow generic expressions can often be used.

43.13.1 Scalar_Source_Kernel_Expression

This Expression assembles a source term for a scalar transport equation. Note that the `MASS` and `SRC` terms can both be cast in this form. The general form is:

$$-m\sum_{g=1}^{N_g}\left(\prod_{r=1}^{N_c}c_r(\boldsymbol{x}_g)\right)\left(\sum_{s=1}^{N_s}f_s(\boldsymbol{x}_g)\right)\phi^i(\boldsymbol{x}_g)|j|w_g \tag{43.3}$$

Here, m is a multiplier that defaults to 1, N_g is the number of Gauss points in the quadrature rule, N_c is the number of coefficients and N_s is the number of sources provided. As an example, consider the term,

$$\int_{\Omega_e}\rho C_p\frac{\partial T}{\partial t}|j|\phi^i\,\mathrm{d}\Omega_e=\sum_{g=1}^{N_g}\rho C_p\frac{\partial T}{\partial t}|j|\phi^i w_g \tag{43.4}$$

In this example, $N_r=2$ with $c_1=\rho$ and $c_2=C_p$; $N_s=1$ with $f_1=\frac{\partial T}{\partial t}$; and $m=-1$. As a side note, in the code the quantities $|j|$ and w_g are treated as additional coefficients for convenience.

43.13.2 Scalar_Advection_Kernel_Expression

This Expression assembles an advection term for a scalar transport equation. The general form is:

$$m\sum_{g=1}^{N_g}\left(\prod_{r=1}^{N_c}c_r(\boldsymbol{x}_g)\right)\boldsymbol{v}_a\cdot\boldsymbol{\nabla}S(\boldsymbol{x}_g)\phi^i(\boldsymbol{x}_g)|j|w_g \tag{43.5}$$

Here, m is a multiplier that defaults to 1, N_g is the number of Gauss points in the quadrature rule, N_c is the number of coefficients, $\boldsymbol{v}_a$ is a configurable advection velocity and S is a scalar field. As an example, consider the term,

$$\int_{\Omega_e} \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T |j| \phi^i \, \mathrm{d}\Omega_e = \sum_{g=1}^{N_g} \rho C_p \boldsymbol{v} \cdot \boldsymbol{\nabla} T |j| \phi^i w_g \tag{43.6}$$

In this example, $N_r = 2$ with $c_1 = \rho$ and $c_2 = C_p$; $S = T$; $\boldsymbol{v}_a = \boldsymbol{v}$; and $m = 1$. As a side note, in the code the quantities $|j|$ and w_g are treated as additional coefficients for convenience.

43.13.3 Scalar_Diffusion_Kernel_Expression

This Expression assembles a diffusion term for a scalar transport equation. The general form is:

$$m \sum_{g=1}^{N_g} \left(\prod_{r=1}^{N_c} c_r(\boldsymbol{x}_g) \right) \left(\sum_{k=1}^{N_F} \boldsymbol{F}_k(\boldsymbol{x}_g) \right) \cdot \boldsymbol{\nabla} \phi^i(\boldsymbol{x}_g) |j| w_g \tag{43.7}$$

Here, m is a multiplier that defaults to 1, N_g is the number of Gauss points in the quadrature rule, N_c is the number of coefficients and N_F is the number of flux models provided. As an example, consider the term,

$$\int_{\Omega_e} \boldsymbol{q} \boldsymbol{\nabla} \phi^i |j| \, \mathrm{d}\Omega_e = \sum_{g=1}^{N_g} \boldsymbol{q} \cdot \boldsymbol{\nabla} \phi^i |j| w_g \tag{43.8}$$

In this example, $N_r = 0$; $N_F = 1$ with $F_1 = \boldsymbol{q}$ where $\boldsymbol{q}$ is the Fourier heat flux, $\boldsymbol{q} = -\kappa \boldsymbol{\nabla} T$; and $m = 1$. As a side note, in the code the quantities $|j|$ and w_g are treated as additional coefficients for convenience.

43.13.4 Vector_Source_Kernel_Expression

This Expression assembles a source term for a vector transport equation. Note that the `MASS` and `SRC` terms can both be cast in this form. The general form is:

$$-m \sum_{g=1}^{N_g} \left(\prod_{r=1}^{N_c} c_r(\boldsymbol{x}_g) \right) \left(\sum_{s=1}^{N_s} \boldsymbol{f}_s(\boldsymbol{x}_g) \right) \phi^i(\boldsymbol{x}_g) |j| w_g \tag{43.9}$$

Here, m is a multiplier that defaults to 1, N_g is the number of Gauss points in the quadrature rule, N_c is the number of coefficients and N_k is the number of sources provided. In this case, the sources f_s are vector quantities. As an example, consider the term,

$$\int_{\Omega_e} \rho \frac{\partial \boldsymbol{v}}{\partial t} |j| \phi^i \, \mathrm{d}\Omega_e = \sum_{g=1}^{N_g} \rho \frac{\partial \boldsymbol{v}}{\partial t} |j| \phi^i w_g \tag{43.10}$$

In this example, $N_1 = 2$ with $c_1 = \rho$, $N_s = 1$ with $f_1 = \frac{\partial \boldsymbol{v}}{\partial t}$; and $m = -1$. As a side note, in the code the quantities $|j|$ and w_g are treated as additional coefficients for convenience.

43.13.5 Vector_Advection_Kernel_Expression

This Expression assembles an advection term for a vector transport equation. The general form is:

$$m \sum_{g=1}^{N_g} \left(\prod_{r=1}^{N_c} c_r(\boldsymbol{x}_g) \right) \boldsymbol{v}_a \cdot \boldsymbol{\nabla} \boldsymbol{V}(\boldsymbol{x}_g) \phi^i(\boldsymbol{x}_g) |j| w_g \tag{43.11}$$

Here, m is a multiplier that defaults to 1, N_g is the number of Gauss points in the quadrature rule, N_c is the number of coefficients, $\boldsymbol{v}_a$ is a configurable advection velocity and $\boldsymbol{V}$ is a vector field. As an example, consider the term,

$$\int_{\Omega_e} \rho \boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{v} |j| \phi^i \, \mathrm{d}\Omega_e = \sum_{g=1}^{N_g} \rho \boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{v} |j| \phi^i w_g \tag{43.12}$$

In this example, $N_r = 1$ with $c_1 = \rho$; $\boldsymbol{V} = \boldsymbol{v}$; $\boldsymbol{v}_a = \boldsymbol{v}$; and $m = 1$. As a side note, in the code the quantities $|j|$ and w_g are treated as additional coefficients for convenience.

43.13.6 Vector_Diffusion_Kernel_Expression

This Expression assembles a diffusion term for a vector transport equation. The general form is:

$$m \sum_{g=1}^{N_g} \left(\prod_{r=1}^{N_c} c_r(\boldsymbol{x}_g) \right) \left(\sum_{k=1}^{N_F} \boldsymbol{F}_k^t(\boldsymbol{x}_g) \right) : \boldsymbol{\nabla} \left(\boldsymbol{e}_j \phi^i(\boldsymbol{x}_g) \right) |j| w_g \tag{43.13}$$

Here, m is a multiplier that defaults to 1, N_g is the number of Gauss points in the quadrature rule, N_c is the number of coefficients and N_F is the number of flux models provided. As an example, consider the term,

$$\int_{\Omega_e} \boldsymbol{T}^t : \boldsymbol{\nabla} \left(\boldsymbol{e}_j \phi^i \right) |j| \, \mathrm{d}\Omega_e = \sum_{g=1}^{N_g} \boldsymbol{T}^t : \boldsymbol{\nabla} \left(\boldsymbol{e}_j \phi^i \right) |j| w_g \tag{43.14}$$

In this example, $N_r = 0$; $N_F = 1$ with $F_1 = \boldsymbol{T}$ where $\boldsymbol{T}$ is the Newtonian stress, $\boldsymbol{T} = \mu \left(\boldsymbol{\nabla} \boldsymbol{v} + \boldsymbol{\nabla} \boldsymbol{v}^t \right) - p \boldsymbol{I}$; and $m = 1$. As a side note, in the code the quantities $|j|$ and w_g are treated as additional coefficients for convenience.

43.14 Defining and Solving ODEs in Aria Plugins

This section covers how to define and solve an ODE or system of ODEs in an Aria plugin. The equations to solve must be of the form,

$$\frac{d\vec{y}}{dt} = f(t, \vec{y}) \tag{43.15}$$

The ODE to solve must be declared as a subclass of the Aria ODE class, and must define at least the derivatives function, and it may also be useful to define the initialization and Jacobian functions:

```
derivatives(double t, const FArray<double,1>& y, FArray<double,1>& dydt)
initialize(double t, const FArray<double,1>& y)
jacobian(double t, const FArray<double,1>& y, FArray<double,2>& dfdy)
```

The initialize() function is called by the ODE solver at construction to set the initial values of $\vec{y}$, and the derivatives() function is called within the solver during solution. The default Jacobian function implements a numerical Jacobian, but it can be overwritten to allow an analytical one.

The currently available solvers are described in the Chemistry Solver Reference chapter. All solvers use adaptive sub-stepping to meet the requested tolerances. The solver stores the problem state (t, $\vec{y}$, and $\frac{d\vec{y}}{dt}$) throughout solution as well as any intermediate states required by the particular solution algorithm. Any of the ODE solvers can be used to solve an ODE that inherits from the Aria ODE class, although for stiff ODEs an appropriate solver should be selected.

43.15 Defining and Using Global Variables

Occasionally it is useful to define a global variable that is available to the user for output to the log or results files, or to be used in user subroutines. These variables may be created using the Region::new_global_variable or Region::new_global_variable2 functions. A variety of value types (e.g. Real, Int, etc.) are supported, as are a number of reduction operations (MPI::Sum, MPI::Max, etc.). Region::new_global_variable() returns a GlobalVariable<Op, T> and is the preferred version. GlobalVariable<Op, T> stores a single value that is globally consistent once it is synchronized using the Region::synchronize_global_variables() call. Region::new_global_variable2() returns a GlobalVariable2<Op, T> which stores 2 values, one local and one global. There are currently some user plugins that access both the local and global values separately so the type is still available for compatibility, but we would prefer to not have both types in the future.

43.16 Errors and Warnings How-To

Editorial note: This How-To is mostly taken from Dave Bauer's collection of How-Tos. Minor changes include formatting, editorial corrections and possibly additional information related to Aria.

43.16.1 Reporting

There are several types of warnings and exceptions that occur in sierra. Warnings are informational messages to the user which may effect the results, but which will allow the execution to complete. Dooms are error conditions which will allow the program to continue to a point, but will not allow it to complete. Often these are within the parser when you want the parser to continue to plod on ahead but not actually execute the code. Exceptions occur when all bets are off.

The Warning and Doom classes send their assembled message to the sierra reporter at destruction. The Exception classes send their assembled message to the sierra reporter within the catch block.

The warning and exception classes will all accept the put-to operator («) with plain old data. Since it is easy to put useful information to the user/developer in the message, please do so. For warnings directed to developers, it is recommended that the message be terminated with a

```
<< std::endl << WarnTrace;
```

For exceptions, always use

```
  << std::endl << StackTrace;
```

`WarnTrace` and `StackTrace` generate a "pretty" source file and line number message. If the message is purely informational for the user the std::endl and WarnTrace for the message should be omitted.

These are the include files:

```
#include <util/Exception.h>// throw sierra
#exception declarations
#include <util/ExceptionReport.h>// sierra reporter and
#RuntimeWarning declarations
```

The `RuntimeUserError` class should be thrown in place of `RuntimeError` when it was the user's fault that the program died. This generally means that the input deck has an error that a `RuntimeDoomed`, the

preferred method since it allows the parser, etc to continue, cannot handle since a seg fault or the like is imminent. It kills the output of the traceback since the user really does care and will only cause confusion.

Also, don't put the `WarnTrace` or `StackTrace` to the `RuntimeUserError`.

The following code snippets serve as examples for the warning, doomed and exception conditions.

```
class_tag *
Parser::prsr_handler_x(
const Prsr_CommandValues & value)
{

    if (runtime_warning_condition)
sierra::RuntimeWarning() << "My useful message about " <<
some_data << std::endl << WarnTrace;


    if (runtime_exception_condition)
        throw sierra::RuntimeError() << "My useful message about " <<
        some_data << std::endl << StackTrace;


    if (runtime_exception_condition)
        throw sierra::RuntimeUserError() << "My useful message about "
        << some_data;


    if (parse_handler_warning_condition)
        sierra::RunWarning() << "My useful message about " <<
        some_data << std::endl << WarnTrace;


    if (parse_handler_doomed_condition)
        sierra::RuntimeDoomed() << "My useful message about " <<
        some_data << std::endl << WarnTrace;

}
```

43.16.2 Throttling a Specific Warning or Doom

If a particular warnings is going to be repeated countless times, you can request a unique message id from sierra using

```
  int message_id = sierra::get_next_message_id();
  int message_id = sierra::get_next_message_id(int max_messages_displayed);
```

and pass `message_id` to the message constructor. This value is often stored within the function that generates the message using a static variable. This technique may not be suitable for all situations. Only a limited number of messages of each id are sent to the sierra reporter.

```
    if (runtime_warning_condition) {
        static message_id = get_next_message_id();
```

```
sierra::RuntimeWarning(message_id) << "My useful message about
        " << some_data << std::endl << WarnTrace;
    }


    if (parse_handler_warning_condition) {
        static message_id = get_next_message_id();
        sierra::Prsr::ParseWarning(value, message_id) << "My useful
        message about " << some_data << std::endl << WarnTrace;
    }


    if (parse_handler_doomed_condition) {
        static message_id = get_next_message_id();
        sierra::Prsr::ParseDoomed(value, message_id) << "My useful
        message about " << some_data << std::endl << WarnTrace;
    }


    if (parse_handler_exception_condition) {
        static message_id = get_next_message_id();
        sierra::Prsr::ParseError(value, message_id) << "My useful
        message about " << some_data << std::endl << WarnTrace;
    }
```

43.16.3 Setting Output Throttles

There are several functions which can throttle the amount of data which is displayed.

To set the maximum number of warnings or dooms before an exception is thrown:

```
    void sierra::set_max_warnings(int max_warnings);
    void sierra::set_max_dooms(int max_dooms);

    int sierra::get_max_warnings();
    int sierra::get_max_dooms();
```

To set the maximum number of warnings displayed. Each occurrence is still counted, but not displayed. Dooms are always displayed.

```
    void sierra::set_max_displayed_warnings(int max_display_warnings);
    int sierra::get_max_displayed_warnings();
```

To set the default maximum id messages to display, set `get_next_message_id()` below:

```
    int sierra::set_default_max_id_dispaly(int
    default_max_id_displayed)
    int sierra::set_default_max_id_dispaly(int
    default_max_id_displayed)
```

43.16.4 Sierra Exception Reporter

What is this sierra reporter thing you ask? It goes like this: When a Warning or a Doom class is destroyed, `report_exception()` is called with the message. `report_exception()` then calls the registered `report_exception_handler`. The `report_exception_handler` has a signature of

```
(*)(const char *message, int type)
```

and is set with `set_report_exception_handler()`. Sierra sets this to use the `sierra_report_exception_handler()` which writes the message to `sierra::Env::output()`.

If you want special decorations around your messages during execution, you can change the reporter. The parse, instantiation and commit handlers are set by `run_sierra()`.

43.16.5 The `Output` versus `OutputP0` Dilemma

Unfortunately, determining what to do with the output from multiple processors can be an issue. If a warning is going to happen on all processors, you may want to wrapper it with a test for processor zero and only output it there. The default `report_exception_handler` sends the output to `Env::output()`

43.16.6 Getting Counts

To get the number of warnings issued, even if not displayed:

```
get_warning_count()
```

To get the number of dooms issued, even if not displayed:

```
get_doomed_count()
```

43.16.7 Traceback and Tracing

The traceback messages are generated by the Trace and Traceback classes upon there destruction. So, if the source code has had trace objects constructed, the stack trace will show them. If efficiency becomes an issue, the `#define app_TRACE_ENABLED` can be undefined which will cause the Trace and Traceback classes to be empty.

The trace object can be used to extract function signature information. The `getFunctionSpec()`, `getFunctionName()`, `getFunctionShortName()`, `getFunctionClass()`, `getFunctionShortClass()` and `getFunctionNamesp` functions will all extract appropriate information from the function signature.

The program `/usr/netpub/traceString/traceString` (on Linux) can be used to quickly place the trace objects in your source code. I suggest editing the results and placing `[ON]`, `[TRACEBACK]` or `[NONE]` after the first `%TRACE` as in `%TRACE[NONE]%`. These will instruct `traceString` which object type to create. Then, rerun `traceString`. I set it to `NONE` for functions which will execute many many times or cannot throw an error or not have much value in a stack trace (accessors, etc). I set it to `TRACEBACK` if it will not be useful to see traced during a trace run, but be useful in a traceback. And `ON` otherwise which incurs some overhead that enables the conditional runtime trace.

Here is the `.traceString` file I use and suggest be used throughout sierra.

```
[START_OF_ROUTINE_PATTERNS]
default   : Trace trace__("$(FQNAME)");
off       :
Off       :
OFF       :
none      :
None      :
NONE      :
spec      : static Tracespec trace__("$(FQNAME)");
Spec      : static Tracespec trace__("$(FQNAME)");
SPEC      : static Tracespec trace__("$(FQNAME)");
on        : Trace trace__("$(FQNAME)");
On        : Trace trace__("$(FQNAME)");
ON        : Trace trace__("$(FQNAME)");
traceback : Traceback trace__("$(FQNAME)");
Traceback : Traceback trace__("$(FQNAME)");
TRACEBACK : Traceback trace__("$(FQNAME)");
```

43.16.8 Deriving from a Sierra Exception

When deriving a new exception from the framework exceptions and using the put-to («) operator, you must include the following template functions in your class or

```
throw MyError() << "My error message cause of " << x;
```

will certainly fail. It only took me a day to relearn this, but:

If `MyError` does not implement a put-to operator, it uses the base classes put to operator, say `RuntimeError`. Well, the `RuntimeError` put-to operator returns a reference to a `RuntimeError`, so now you are going to be throwing a `RuntimeError` exception not a `MyError` exception. This is only an issue when using the temporaries. By putting the `MyError()` construction on the same line as the throw. I.e.,

```
MyError x;
x << "My error message cause of " << x;
throw x;
```

works fine since we are actually throwing `x`, a `MyError` object.

So, add these if you derive from a sierra exception and wish to throw the temporary object and use the put operator all in one pretty line:

```
class MyError : public RuntimeError {

.
.
.
  inline MyError& operator<<(ExceptionString& (*f)(ExceptionString
&)) {
    f(*this);
    return *this;
  }
```

```
  template <class T>
  inline MyError &operator<<(const T &t) {
    RuntimeError::operator<<(t);
    return *this;
  }
};
```

43.16.9 path_name()

When generating error messages with object names, use the `path_name()` form rather than just name. This generates a dot (`.`) separated list of names from the `Procedure` on down.

43.16.10 abort() – Don't use it

`abort()` does not provide any useful information when the application dies. By replacing abort calls with Warnings, Dooms and Exceptions, the code will be easier to maintain and the user will get better diagnostic output.

43.16.11 Apub_Parser_Base – Useful for parsing, not needed for error reporting

This class stashes useful information from within a parser that is likely to be needed after parsing. The runtime error reporting routines handle error reporting making the line number and command value information redundant for error reporting.

43.17 Diagnostic Writer How-To

Editorial note: This How-To is mostly taken from Dave Bauer's collection of How-Tos. Minor changes include formatting, editorial corrections and possibly additional information related to Aria.

This document briefly describes the implementation of the diagnostic writer and masked based output. The diagnostic writer is intended to replace debug level output with but masked based diagnostic output. By utilizing the diagnostic writer, your normal output can be separated or interleaved with the diagnostic output, the diagnostic output can the enabled/disabled at specific times during a run, and entire classes can be output by simply putting the object to the diagnostic writer.

43.17.1 Output

Application output falls into three classes. Normal execution output, warning and error output, and diagnostic output. Normal execution output is handled via the `Env::outputP0()` and `Env::output()` functions and by the application diagnostic writer. Warning and error output is handled by the `RuntimeWarning`, `RuntimeDoomed` and runtime exception classes (`RuntimeError`, `LogicError`, etc), which is eventually written to the env output stream and the diagnostic output stream. And diagnostic output is for selected operationally descriptive output.

In many cases, the diagnostic output is utilized as the primary output stream when selectable level of user output is desired. With that in mind, an InfoWriter class has been written. However, additional discussion is required to complete the design and implementation of this class for primary application output.

43.17.2 Diagnostic Writer

The diagnostic writer allows you to write diagnostic information to the diagnostics stream by specifying the content from the command line or the input deck. It a debug level built on a bit mask.

Since there are many applications and libraries, there are several diagnostic writers. Each diagnostic writer has its own bit mask, command line parser and writer. However, they all share a common diagnostic stream. So, output from each diagnostic writer is properly interleaved.

Each application defines it own diagnostic writer. This is generally defined within the `app_DiagWriter_fwd.h`, `app_DiagWriter.h`, `app_DiagWriter.C` files. The `app_DiagWriter_fwd.h` file defines the `LOG_xxx` bit assignments. These values are used to specify the type of message to be written. The `app_DiagWriter.h` rarely needs to be modified. It declares the diagnostic writer for the application or library. The `app_DiagWriter.C` files defines the parser which provides names for the `LOG_xxx bit` masks.

43.17.3 Using The Diagnostic Writer

To have your program send output whenever a specified log bit is set, add the following to your code:

```
dwout.m(LOG_xxx) << "description, " << value << std::endl;
```

or, if much computation or MPI is involved:

```
if (dwout.shouldPrint(LOG_xxx)) {
  dout << "really_spendy_function(), " <<
  really_spendy_function() << std::endl;
}
```

Where `dwout` is the name of the diagnostic writer, `LOG_xxx` is the bit which describes the type of message, `description` is a description of the data. value is the value of interest. And, `std::endl` ends the message.

Note that nearly all (please let me know what's missing) containers have output operators. So to write an entire vector, just write the vector variable, the diagnostic writer will take case of the rest.

43.17.4 Turning on the LOG_xxx Bits

Each application has its own command line option and line commands for flipping on the bits. The command line option has limited functionality in that the parameters cannot be switched on and off during an application execution. However, it is useful for a quick look.

For sierra framework, the command line option is `-m`, and each application has it's own option name. Use the `-h` option to display a table of command line options and parameters.

The `Diagnostic Control` command block in the sierra block controls each the diagnostic writers:

```
Begin Diagnostic Control <diagwriter>
   From time t0 to t1 enable <parameters>
   From step s0 to s1 enable <parameters>
   On condition c enable <parameters>
   Enable <parameters>
End Diagnostic Control <diagwriter>
```

Option	Description
`error`	Display error messages
`warning`	Display warning messages
`members`	Display data structure members
`timer`	Display execution time during trace
`trace`	Display execution trace
`contact`	Display contact diagnostic information
`geometry`	Display geometry diagnostic information
`scontrol`	Display solver control diagnostic information
`search`	Display search diagnostic information
`transfer`	Display transfer diagnostic information
`parser`	Display parser diagnostic information
`parameters`	Display parameter diagnostic information
`io`	Display contact I/O information
`plugins`	Display user function and plugin diagnostic information

Table 43.1. Diagnostic writer options for Framework (`fmwkout`).

Option	Description
`bc`	Display boundary condition information
`debug`	Display extra debugging information
`eq`	Display equation information
`expression`	Display Expression information
`hadapt`	Display h-adaptivity information
`linsolve`	Display linear solver information
`nonlinear`	Display nonlinear solver information
`pp`	Display postprocessor information
`sens_check`	Enable the Expression Newton sensitivity checker
`species`	Display species information
`transfer`	Display transfer related information
`plugin`	Display plugin information

Table 43.2. Diagnostic writer options for Aria (`arialog`).

Please refer to Diagnostic Control for implementation details.

The diagnostic output can be selectively enabled based on time, step or an application specified condition. During the application's procedure execution loop, the diagnostic controller evaluates the enclosed line commands in the order specified in the input deck. The diagnostic options specified in the first line command that meets its criteria are applied.

Since control parameters are only applied when the criteria is met, it is important to include an `ENABLE` line command with the base settings to be applied as a baseline.

Table 43.17.4 lists some of the options available for the framework diagnostic writer `fmwkout` and options for the Aria diagnostic writer `arialog` are given in table 43.17.4. Other application will likely implement additional diagnostic writers.

43.17.5 Diagnostic Stream

The diagnostic stream specified the output destination for all the active diagnostic writers. The stream allows the output from the diagnostic writers to be interleaved.

The command line option

```
-dout <destination>
```

and the sierra block line command

```
diagnostic stream <destination>
```

determines the output. `<destination>` may be `cout`, `cerr`, `outputp0`, `output` or a path. If a path is given, each processor creates its own file suffixing the path with `.n.p` where `n` is the number of processors and `p` is the rank of each processor. See Diagnostic Stream for specifying the output destination.

43.17.6 Coding Objects

To code your own objects to play with the diagnostic writer, you only need to add a `verbose_print()` member function to your class. And, a `operator«()` function to your namespace. Naturally the implementation for `verbose_print()` is generally in the `.C` file, not the header.

```
class MyClass : public ParentClass
{
  public:
  DiagWriter &verbose_print(DiagWriter &dout) const {
    if (dout.shouldPrint()) {
dout << "MyClass " << m_name << push << std::endl;
ParentClass::verbose_print(dout).std::endl();
dout << "m_var1, " << m_var1 << std::endl;
dout << "m_var2, " << m_var2 << std::endl;
dout << "m_ptr1, " << c_ptr(m_ptr1) << std::endl;
dout << "m_ptr2, " << c_ptr_name(m_ptr2) << std::endl;
dout << pop;
     }
     return dout;
  }
};

inline DiagWriter &operator<<(Diagwriter &dout, const MyClass
&my_class) {
  return my_class.verbose_print(dout);
}
```

If your class is polymorphic, be sure to define `verbose_print()` as `virtual`.

When writing a subclass from your class, the put-to operator (`«`) will by do what you expect, even when you cast. So you need to use the direct call form.

43.17.7 Writing Containers

You can write an STL container by simple putting it to the diagnostic writer. This is implemented using templates in `utility/include/DiagWriter.h`.

43.17.8 Writing Pointers

Writing of pointers is usually quite ugly since you need to check if the pointer is null first. Instead, use the `c_ptr()` function. If the pointer is null it writes `"(pointer), <not created>"`. Or, you can use `c_ptr_name()` function which will call the `name()` function of the pointed to object if the pointer is not null.

You can have you own pointed object function called by replacing name with your member function name below.

```
template <class T>
inline c_ptr_func_<T, const String &> c_ptr_name(const T *t) {
  return c_ptr_func_<T, const String &>(t, &T::name);
}
```

43.17.9 Diagnostic Control

The diagnostic control block in the sierra command block is handled by the `Fmwk::DiagControl::Control` class. Simply add

```
Fmwk::DiagControl::Control diag_control(step_cntr(),
                                        time(Fmwk::STATE_OLD), 0);
```

at the beginning of your main procedure control loop. This line exists in the Solver Control procedure.

43.18 Timers and Timing How-To

Editorial note: This How-To is mostly taken from Dave Bauer's collection of How-Tos. Minor changes include formatting, editorial corrections and possibly additional information related to Aria.

This document briefly describes the time metrics collection features available to sierra applications.

The `DiagTimer` and `Timer` classes provide runtime metric information for properly rigged objects.

The system has a root "System" timer. This timer is started when `run_sierra()` is called and stopped before the successful completion information is displayed.

43.18.1 DiagTimer and Timer

The `DiagTimer` class implements the basic developer interface to the timers. Generally, the framework form, `Timer`, will be used to implement timers within a framework derived class.

Timers are intended to be members of your classes or static objects created within a function or member function. Each timer has a name, a parent and a timer class. The name is used to find a child timer of a

parent and to display the collected metrics. The parent is used to build the hierarchy of metrics gathered in a application. And the timer class or type categorizes the timers so they may be enabled, disabled and selected for display.

The timing information is actually maintained in a separate tree. So, during timer construction within a class, the timer is a reference to the real timing metric information in the tree. This design means that timers are not destroyed when an object is destroyed.

The timers are hierarchical by name. So, by giving a timer a unique name among its siblings, each object has it's own timer and it's children timers are also unique.

43.18.2 Add a Timer to a Class

To add a timer to a class, include the header file and add the timer as a member of the class. Then, during construction initialization, specify the timer's name and parent timer. You can also specify a timer class if it is different from the parents.

Then, to start/stop the timer, use the TimeBlock and TimeBlockSynchronized to ensure that a started timer is stopped.

```
#include <util/Timer.h>

class MyClass
{
public:
    // etc.

    Timer &getMyTimer() {
        return m_myTimer;
    }

    Timer &getMySubTimer() {
        return m_mySubTimer;
    }

private:

    Timer m_myTimer;
    Timer m_mySubTimer;
    // etc.
};


MyClass::MyClass(
    Region & region)
  : m_myTimer("My Timer", region.getRegionTimer()),
    m_mySubTimer("My SubTimer", m_myTimer)
{
    // etc.
}


// When controlling timers using the TimeBlocks, be sure to give the
```

```
// object a name.  Some compilers destroy unnamed objects immediately,
// other destroy them at the end of the block. I usually use timer__

MyClass::someFunction()
{
    Timer::TimeBlock timer__(m_myTimer); // Metrics in block are
 //   collected to m_myTimer;
}


MyClass::someFunction()
{
    Timer::TimeBlockSynchronized timer__(m_myTimer);
//  MPI_Barrier then start timer
}


MyClass::someOtherFunction()
{
    m_myTimer.start(); // Works, but is dangerous if stop() is not called.
    m_myTimer.stop();
}
```

43.18.3 Adding a Timer to a Function

To add a timer to a function, create a static Timer in the function, then start/stop it using the Timer::TimeBlock. Note that if there is no parent specified, the root timer "System" will be the timer's parent.

```
int
myFunction()
{
    static Timer my_timer("My Timer");
    Timer::TimerBlock(my_timer);
    // etc.
}
```

43.18.4 Getting Information from a Timer

Each timer has an accumulation time/count, a checkpoint time/count and a recent lap time/count. The accumulation time is the overall time/count since the start of the application. The checkpoint time/count records the current values when set, then the difference from that time/count can be obtained. This is useful for displaying delta times for iterations. The lap time is the time accumulated during the last `start()`/`stop()` cycle for the timer.

The metrics available are `getCPUTime()`, `getWallTime()`, `getFlopCount()`, `getIOCount()`, `getMsgCount()`. Flop, IO, and Msg are not implemented on most platforms (any really).

```
m_myTimer.getCpuTime().getStart(); // Timer most recent start time
m_myTimer.getCpuTime().getStop(); // Timer most recent stop time
m_myTimer.getCpuTime().getLap(); // Most recent stop - start
```

```
m_myTimer.getCpuTime().getTotal(); // Accumulated time
m_myTimer.getCpuTime().getTotal(false); // Accumulated time less checkpoint time
```

I could add `getCheckpoint()` and `getCheckpointTotal()` as variations on a theme, but...

43.18.5 Displaying the Timers

To display a table of the collected timing information, use the `Timer::printTable()` function. The function lets you specify the classes and the metrics to output. Note that only enabled timers and metrics are displayed. Use `TIMER_CHECKPOINT` to display checkpointed time. It also resets the checkpoint time/count after display.

```
std::cout << Timer::printTable(TIMER_CPU | TIMER_ALL);
```

43.18.6 Enabling/Disabling the Timers

Timers are enabled by timer class. They can be enabled using the `Timer::setEnabledTimers()` function, by the `-timer` command line option or by the `ENABLE TIMER` input deck line command.

43.18.7 Timers in the Framework

The framework creates several timers and starts/stops them when it has control of the operations.

header timers are given the name associated with the name of the object automatic timers are handled within framework and required no additional coding otherwise, instructions are given on how to collect the data for the timer

```
Domain:
  Domain                  header
  LinearSystem            automatic
  Initialize              automatic
  Execute                 automatic
  Load                    automatic
  Solve                   automatic

  Procedure:
    Procedure             header
    Initialize            automatic
    Restart               automatic
    Execute               automatic
    MeshInput             automatic
    MeshOutput            automatic
    Transfer              automatic

    Region:
      Region              header
      Initialize          call Timer::TimeBlock timer__(getRegionInitializeTimer()); at beginning of in
      Execute             call Timer::TimeBlock timer__(getRegionExecuteTimer()); at beginning of execu
```

```
    Mechanics:
      Mechanics         header

      Algorithm:
        Algorithm       header
        Apply           automatic

      WorksetAlgorithm: (sub class of Algorithm)
        Gather          automatic
        ScatterAssemble automatic

    NonLinearCoupler:
      NonLinearSolver   automatic
      Initialize        automatic
      Scatter           automatic
      Solve             automatic

    NonLinearSolver:
      NonLinearSolve    automatic
      Initialize        automatic
      Scatter           automatic
      Solve             automatic

UserInputFunction:
  UserInputFunction     automatic
```

Glossary

coefficient: Each *field* that is represented by a basis function expansion has a set of coefficients that are used in that expansion. For example, the three dimensional velocity vector represented by an eight node, tri-linear hex element has eight coefficients. In this example, each coefficient is also a vector with three *components*.

This is consistent with the Sierra data model wherein vectors and tensors are single-entity data types.

Although for certain basis function representations the coefficients may be the exact value of the *field* at a point this is not the case in general.

component: The number of values required to describe a *field* at a point. Equal to the tensor dimension raised to the power of the tensor order. For example, temperature has one component, velocity has 3 components (in 3 dimensions) and stress has 9 components.

dof: An entry in the vector of unknowns, i.e. , in the linear solver solution vector.

field: The physical variable of interest, e.g. , temperature or velocity.

multidof: A *field* is a multidof if it contains more than one *component*

References

[1] Micheal W. Glass. Computational Simulation home page. http://compsim.sandia.gov/compsim/. 1.1

[2] Patrick K. Notz. Thermal/Fluid Developer's Trac Wiki. https://prod.sandia.gov/sierra-trac/wiki/Modules/Aria. 1.1

[3] The SNTools Project. SNTools SourceForge Project. Online. 2.3

[4] Javier Bonet and Richard D. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis.* Cambridge University Press, 1997. 4.30.3, 4.45.3, 5.16, 1

[5] Ted Belytschko, Wing Kam Liu, and Brian Moran. *Nonlinear Finite Elements for Continua and Structures.* John Wiley and Sons, 2004. 4.30.3, 4.45.3, 5.16, 1, 2

[6] I. M. Krieger. Rheology of monodisperse latices. *Adv. Colloid Interface Sci.*, 3:111–136, 1972. 4.56.10

[7] W. M. Deen. *Analysis of Transport Phenomena.* Topics in Chemical Engineering. Oxford University Press, New York, 1998. 5.1

[8] M. J. Martinez. Mathematical and numerical formulation of nonisothermal multicomponent three-phase flow in porous media. Technical Report SAND95-1247, Sandia National Laboratories, Albuquerque, NM, USA, 1995. http://www.prod.sandia.gov/cgi-bin/techlib/access-control.pl/1995/951247.pdf. 5.11.1

[9] M. J. Martinez, P. L. Hopkins, and J. N. Shadid. LDRD final report: physical simulation of nonisothermal multiphase multicomponent flow in porous media. Technical Report SAND97-1766, Sandia National Laboratories, Albuquerque, NM, USA, 1997. http://www.prod.sandia.gov/cgi-bin/techlib/access-control.pl/1997/971766.pdf. 5.11.1

[10] M. J. Martinez, P. L. Hopkins, and P. N. Reeves. PorSalsa user's manual. Technical Report SAND2001-1555, Sandia National Laboratories, Albuquerque, NM, USA, 2001. http://www.prod.sandia.gov/cgi-bin/techlib/access-control.pl/2001/011555.pdf. 5.11.1

[11] Chris Lautenberger and Carlos Fernandez-Pello. Generalized pyrolysis model for combustible solids. *Fire Safety Journal*, 44(6):819–839, 2009. 5.11.2

[12] D. K. Gartling, C. E. Hickox, and R. C. Givler. Simulations of coupled viscous and porous flow problems. *Comp. Fluid Dyn.*, 1-2:23–48, 1997. 5.12

[13] S. A. Roberts, D. R. Noble, E. M. Benner, and P. R. Schunk. Multiphase hydrodynamic lubrication flow using a three-dimensional shell finite element model. *Comput. Fluids*, 2013. , in press. 5.13

[14] Lawrence E. Malvern. *Introduction to the Mechanics of a Continuous Medium.* Series in Engineering of the Physical Sciences. Prentice-Hall, Upper Saddle River, NJ, USA, 1969. 5.16, 1, 3

[15] George E. Mase. *Theory and Problems of Continuum Mechanics.* Schaum's Outline Series. McGraw-Hill, New York, NY, USA, 1970. 5.16, 1, 3, 4, 6, 7

[16] Clark R. Dohrmann and Pavel B. Bochev. A stabilized finite element method for the Stokes problem based on polynomial pressure projections. *Int. J. Num. Meth. Fluids*, 46:183–201, 2004. 6.23

[17] Pavel B. Bochev, Clark R. Rohrmann, and Max D. Gunzburger. Stabilization of low-order mixed finite elements for the Stokes equations. *SIAM J. Numer. Anal.*, 44(1):82–101, 2006. 6.23

[18] T. Hughes, L. P. Franca, and M. Balestra. A new finite element formulation for computational fluid dynamics: V. circumventing the babuska-brezzi condition: a stable petrov-galerkin formulation of the stokes problem accomodating equal-order interpolations. *Comput. Methods Appl. Mech. Engrg.*, 59:85–99, 1986. 6.23

[19] R. A. Cairncross, P. R. Schunk, T. A. Baer, R. R. Rao, and P. A. Sackinger. A finite element method for free surface flows of incompressible fluids in three dimensions. part i. boundary fitted mesh motion. *Int. J. Numer. Methd. Fluids*, 33:375–403, 2000. 9.2.22

[20] T. D. Blake and J. De Coninck. The influence of solid-liquid interactions on dynamic wetting. *Adv. Colloid and Interface Sci.*, 96:21–36, 2002. 9.2.34

[21] G. S. Beavers and D. D. Joseph. Boundary conditions at a naturally permeable wall. *J. Fluid Mech.*, 30:197–207, 1967. 9.5

[22] P. G. Saffman. On the boundary condition at the surface of a porous medium. *Studies in Applied Mathematics*, 50(2):93–101, 1971. 9.5, 9.5

[23] R. H. Davis and H. A. Stone. Flow through beds of porous particles. *Chemical Engineering Science*, 48(23):3993–4005, 1993. 9.5

[24] Ken S. Chen, Gregory H. Evens, Richard S. Larson, David R. Noble, and William G. Houf. Final report on LDRD project: A phenomenological model for multicomponent transport with simultaneous electrochemical reactions in concentrated solutions. SAND 2000-0207, Sandia National Laboratories, Albuquerque, NM 87185, USA, January 2000. 11.3.15

[25] I. M. Levi. Users' Guide to the Transient Heat Conduction Finite Element Code HTCON. Technical Report MM70-5425-17, Bell Telephone Laboratories, Whippany, NJ, USA, 1970. 16.2

[26] A. F. Emery, K. Sugihara, and A. T. Jones. A comparison of some of the thermal characteristics of finite-element and finite-difference calculations of transient problems. *Num. Heat Trans.*, 2:97–113, 1979. 16.2

[27] F. Damjanic and D. R. J. Owen. Practical considerations for thermal transient finite element analysis using isoparametric elements. *Nucl. Engrg. Des.*, 69:109–126, 1982. 16.2

[28] E. Rank, C. Katz, and H. Werner. On the importance of the discrete maximum principle in transient analysis using finite element methods. *Int. J. Num. Meth. Engng.*, 19:1771–1782, 1983. 16.2

[29] H. S. Carslaw and J. C. Jaeger. *Conduction of Heat in Solids*. Clarendon Press, Oxford, 2nd edition, 1959. 16.2

[30] D. K. Gartling. NACHOS II - a finite element computer program for incompressible flow problems. part I - theoretical background. Technical Report SAND86-1816, Sandia National Labs, Albuquerque, NM, USA, April 1986. 16.5.26

[31] Erik Boman, Karen Devine, Robert Heaphy, Bruce Hendrickson, William Mitchell, Matthew St. John, and Courtenay Vaughan. Zoltan home page. http://www.cs.sandia.gov/Zoltan, 1999. 19

[32] Karen Devine, Bruce Hendrickson, Erik Boman, Matthew St. John, and Courtenay Vaughan. *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1377 http://www.cs.sandia.gov/Zoltan/ug_html/ug.html. 19

[33] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002. 19

[34] Kevin D. Copps and Brian C. Carnes. Sierra Verification Module: Encore User Guide - Version 4.28. SAND 2013-3344, Sandia National Laboratories, Albuquerque, NM 87185, USA, June 2013. 21.1

[35] R. Siegel and J. R. Howell. *Thermal Radiation Heat Transfer*. Taylor and Francis, Washington, D.C., 1992. 22.1

[36] M. W. Glass. Chaparral: A library for solving enclosure radiation heat transfer problems. Technical Report SAND95-2049, Sandia National Laboratories, Albuquerque, New Mexico, 1995. 22.2

[37] Robert Kosik, Peter Fleischmann, Bernhard Haindl, Paola Pietra, and Siegfried Selberherr. On the interplay between meshing and discretization in three-dimensional diffusion simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(11):1233–1240, 2000. 33.3

[38] Changna Lu, Weizhang Huang, and Jianxian Qiu. Maximum principle in linear finite element approximations of anisotropic diffusion–convection–reaction problems. *Numerische Mathematik*, 127(3):515–537, 2014. 33.3

[39] Dmitri Kuzmin. Linearity-preserving flux correction and convergence acceleration for constrained galerkin schemes. *Journal of Computational and Applied Mathematics*, 236(9):2317–2337, 2012. 33.3

[40] Dmitri Kuzmina and John N Shadidb. A new approach to enforcing discrete maximum principles in continuous galerkin methods for convection-dominated transport equations. *Preprint: Ergebnisberichte des Instituts für Angewandte Mathematik*, 529. 33.3

[41] E. R. G. Eckert. Survey of boundary layer heat transfer at high velocities and high temperatures. Technical report, WADC, 1960. 33.10

[42] D. Dobranich. Safsim input manual -a compute program for the engineering simulation of flow systems. Technical Report SAND92-0694, Sandia National Laboratories, Albuquerque, NM, USA, September 1992. 34.1

[43] I. Glassman. *Combustion*. Academic Press, New York, 1977. 1

[44] S. Gordan and B. J. McBride. Computer program for calculation of complex chemical equilibrium compositions and applications I. analysis. Technical Report Ref. Publication 1311, NASA, October 1994. 41.5.2

[45] B. J. McBride and S. Gordan. Computer program for calculation of complex chemical equilibrium compositions and applications II. users manual and program description. Technical Report Ref. Publication 1311, NASA, June 1996. 41.5.2

[46] P. K. Notz, S. R. Subia, M. H. Hopkins, and P. A. Sackinger. A novel approach to solving highly coupled equations in a dynamic, extensible and efficient way. In M. Papadrakakis, E. Onate, and B. Schrefler, editors, *Computation Methods for Coupled Problems in Science and Engineering*, page 129, Barcelona, Spain, April 2005. Intl. Center for Num. Meth. in Engng. (CIMNE). 43.2

[47] Bjarne Stroustrup. *The C++ Programming Language, Special Edition*. Addison Wesley, Reading, MA, 2000. 43.11.1

Index

B

C

D

E

F

G

H

I

J

K

N

O

R

S

T

U

V

W

X

Y

Z

DISTRIBUTION:

1 MS 0899 Technical Library, 9536 (electronic copy)


Sandia National Laboratories