SANDIA REPORT

SAND2016-10194
Unlimited Release
Printed October 10, 2016

SIERRA Verification Module: Encore User Guide – Version 4.42

Kevin D. Copps and Brian R. Carnes

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

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

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

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

SAND2016-10194
Unlimited Release
Printed October 10, 2016

SIERRA Verification Module: Encore User Guide – Version 4.42

Kevin D. Copps and Brian R. Carnes

Sandia National Laboratories
Validation and Uncertainty Quantification Processes, Dept. 1544
P.O. Box 5800
Albuquerque, NM 87185-0828
kdcopps@sandia.gov
bcarnes@sandia.gov

Abstract

The *Encore* software package is both a stand-alone application and a software library. This guide explains the syntax of Encore input, provides examples, and is a comprehensive catalog of the Encore commands. Acting as a stand-alone application, Encore provides utilities for reading solutions from files and enables solution verification, postprocessing, field transfers, and basic mesh refinement. Acting as a software library, Encore is a component of the fluid, thermal, and solid modeling applications in the Sierra Mechanics suite. As a library, Encore provides the enclosing modeling application a superset of the stand-alone capabilities—enabled by application specific information—including physics specific postprocessors and adaptive mesh refinement.

Contents

Preface

Goals of this Manual

In this volume we describe the software package called Encore and how to use it. We help you construct an input file for either *offline use* of the Encore application, or *online use* of Encore capabilities as part of one of the Sierra Mechanics analysis codes, such as Adagio, Aria, or Calore. Encore is being developed at Sandia National Laboratories under the ASC (Advanced Simulation and Computing) program of the NNSA (National Nuclear Security Administration).

The current version of this document provides a comprehensive, and up to date reference for the valid syntax of Encore capabilities. This manual, and the Encore software package, should be useful to anyone who either uses Sierra Mechanics applications, or has a need to postprocess analysis results in the ExodusII format. This manual does not require you to have previous experience with Sierra input command syntax.

Changes Since Earlier Editions

Readers familiar with earlier editions, which were entitled *Encore Reference Manual*, may notice a number of changes. We have changed the name to *Encore User Guide*, partly to reflect the additional content. With Sierra version 4.28 and later, we follow a standard Sierra naming convention for our manuals which includes the Sierra version number. We have kept the comprehensive reference for all possible Encore commands grouped into chapters by related functionality. But we have added new helpful explanations and descriptions to the start of most chapters. And we have incorporated many references to working example input files. The example input files are listed in the Appendix. We tried to make the examples representative of actual analysis activities while illustrating complex use cases, as well as illustrating a few simpler use cases.

Updates in Sierra version 4.28

Starting in Sierra version 4.28, we've documented a few new postprocessors. Both the Realize Random Field postprocessor, in section 2.13, and the Realize PC Random Field, in section 2.7, can be used to compute realizations of random fields. The Sierra KL Solver in section 2.22 computes eigenvalues/eigenvectors for a discrete Karhunen-Loève solve given a covariance func- tion, and a related command block is the KL Solver Error Indicator in section 2.6.

Updates in Sierra version 4.34

Based on feedback from thermal analysts, three changes were made to the Evaluate Function Postprocessor (also referred to as "special points" by the Aria analysis community), in

section 2.10.

1. Abort whenever an evaluation point is not found within the parametric tolerance. This is intended to prevent an analysis to proceed when points lie well outside the mesh. However, this should only happen once (even if multiple points fail). The log file will indicate the results of the search for each point and will suggest a new tolerance that should be large enough to locate the point near an element in the mesh. The analysis can then be re-run with larger tolerances, the failed point locations can be adjusted, or the failed points can be removed.

2. The line command `Fail Action Is ...` has been removed, since it no longer has any functionality. This will require users to remove this line command from any analysis input files for *Aria*, *Arpeggio*, and *Encore*.

3. When a point is found outside an element but within the parametric tolerance, instead of extrapolating outside the element, and possibly introducing non-physical values, the point will be projected to the nearest point within the element. The log file will indicate this has happened. This is currently implemented for linear tri, tet, quad, and hex elements only.

Organization of this Manual

There is no best way to approach the chapters in this manual, the material does not lend itself to a complete linear progression. Thus, you may find yourself skipping back and forth because many commands in Encore have related dependencies to other commands. Nevertheless we have tried to place the information unique to Encore close to the beginning, and save the more generic Sierra commands for the end. Although the latter chapters discuss indispensable capabilities required for any Encore input, these chapters are generally covering capabilities common to all of the applications in the Sierra Mechanics suite.

In the Introduction, we first describe the conventions used in this manual, and second we lay out the format and structure of an Encore input file. Next, in chapter 1, we introduce the foundation of most of Encore's pre- and postprocessor capabilities: the concept of Functions. Following that, is chapter 2 discussing Postprocessors and Norms, which are the many ways in which Encore computes calculations involving *Functions*.

The middle chapters each focus on a set of commands organized by the kind of special Encore capability they provide. The first is chapter 3 on field Transfers. Followed by chapter 4 and chapter 5, on the topics of Indicators and Markers, respectively.

The next two, chapter 6 and chapter 7 of the manual focus on outlining commands that are used at different levels of the Sierra hierarchy, respectively the Domain, Procedure, and Region level. The larger of these, chapter 7, actually goes through the many lower level commands available at the Region level, many of which may have been referred to in the previous chapters.

The chapter 8 is on the topic of Solution Control, which is the general way for Sierra applications to control the transient simulation, or order of *events* during execution of the code. Solution control is also used to control loosely coupled multiphysics. By *events*, we mean those calculations or procedures that are order dependent, e.g., solves, transfers, postprocessors, indicators, markers, and mesh refinement.

The next chapter 9 focuses on Input and Output of mesh information and the *heartbeat* and *history output*.

For late breaking changes or additions to material in this manual, you may also want to review any release notes for the specific version of Encore you are using, which are distributed with the software.

How To Contact Us

Please address comments and questions concerning this manual to:

Kevin Copps
kdcopps@sandia.gov
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0828
505-844-4521 (voice)
505-284-2418 (fax)

Brian Carnes
bcarnes@sandia.gov
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0828
505-284-1332 (voice)
505-844-4523 (fax)

For more information on Encore, visit the website at http://compsim.sandia.gov.

Acknowledgements

We would like to thank Christopher Newman and David Neckels, who implemented earlier Sierra Framework and Calore based versions of some the capabilities now in Encore. We would like extend special thanks to Derek Gaston, who devoted tremendous energy to the first versions of Encore, and implemented its overall structure. We also extend our gratitude to all those responsible for developing and improving the Sierra system in the past and present. We would especially like to thank those who, without their help, the Encore package would not be possible: David Baur, Steve Bova, Nathan Crane, Stefan Domino, Carter Edwards, Mike Glass, Mark Hamilton, Erik Ilescas, James Overfelt, Elijah Newren, Pat Notz, Kendall Pierson, Greg Sjaardema, Samuel Subia, and Alan Williams.

— Kevin Copps & Brian Carnes
Albuquerque, New Mexico
April 2013

Introduction

Background

Encore is a component of the Sierra Mechanics package. The primary goal of the Encore software component is to enable predictive simulation. This means providing tools so that developers may verify the accuracy of a numerical code, and analysts can control the numerical error in the approximation. We distinguish between two types of verification:

code verification: which ensures that numerical algorithms are free of bugs, and usually takes the form of testing the order-of-convergence to an exact solution when the element size and time-steps are reduced; and

solution verification: which ensures that the a numerical approximation to a real world problem has converged to an answer within a desired tolerance, and this often involves managing the tradeoff of computer/human resources versus accuracy.

For both types of verification, Encore provides a toolkit to help you accomplish your task. You may be able to find further information on Encore, not necessarily contained in this manual at http://compsim.sandia.gov.

Capabilities and Features

The capabilities and features of the Encore component can be separated into four related groups.

1. **Pre- And Post-Processing:**
   - Calculate derived and integrated responses from the numerical approximation.
   - Higher order smoothing, or recovery, of mesh fields, e.g., construct a linear field from a piecewise constant.
   - Transfer, or map, mesh fields from one mesh to another mesh, on volumes, or surfaces.
   - Extendable functionality through user written plug-ins, or subroutines (C++ or Fortran).
2. **Error Metrics:**
   - Generate and drive exact analytical solutions, or *manufactured* solutions.
   - Compute norms of solution fields and their derivatives with respect to analytical or other solution fields.

3. **Error Estimation:**
    - Compute various mechanics independent error indicators (a relative contribution to error on each mesh element).
    - Compute an estimate of the discretization error for a derived response, or observable, quantity of interest.
4. **Adaptivity And Mesh Refinement:**
    - Refine and un-refine sets of elements (with *hanging-nodes*); elements of type hexahedral, tetrahedral, quadrilateral, triangular and shells, etc.
    - Mark elements based on geometry, field values, or error indicators/estimators, with limits on total mesh size.

The source code of Encore is written in C++ and is currently around forty-thousand lines. The project development proceeds by a small team of two or three people. The test suite for Encore itself contains more than 180 tests, each testing different functionality.

Running Encore

You can run Encore commands in two ways.

Software library: Encore is a software library that can be statically linked with any Sierra Mechanics application, thus offering many of capabilities directly within the execution of a mechanics model.

Standalone executable: Encore can be executed from the command line, taking two types of inputs, 1) one or more meshes in the form of a ExodusII databases, and 2) a single textual input file (Sierra syntax).

The Encore executable is installed as part of the regular installation of Sierra Mechanics. Running the Encore executable is similar to running any other Sierra application—it is done using the command line "`sierra`" script. The `sierra` script takes care of a lot of things for you, for example, the submission to a parallel queue on a shared supercomputer, and the initial parallel decompositions of your mesh. Typically, to run an Encore input file you would execute the following on the command line:

```
sierra encore -i myinputfile.i
```

This runs Encore, passing in the input file `myinputfile.i`. There are hundreds of options to the sierra command, use `sierra --help`, `sierra encore --help`, or see the relevant web pages for further options. And if you can visit the website containing all the documentation for Sierra and other tools at http://compsim.sandia.gov.

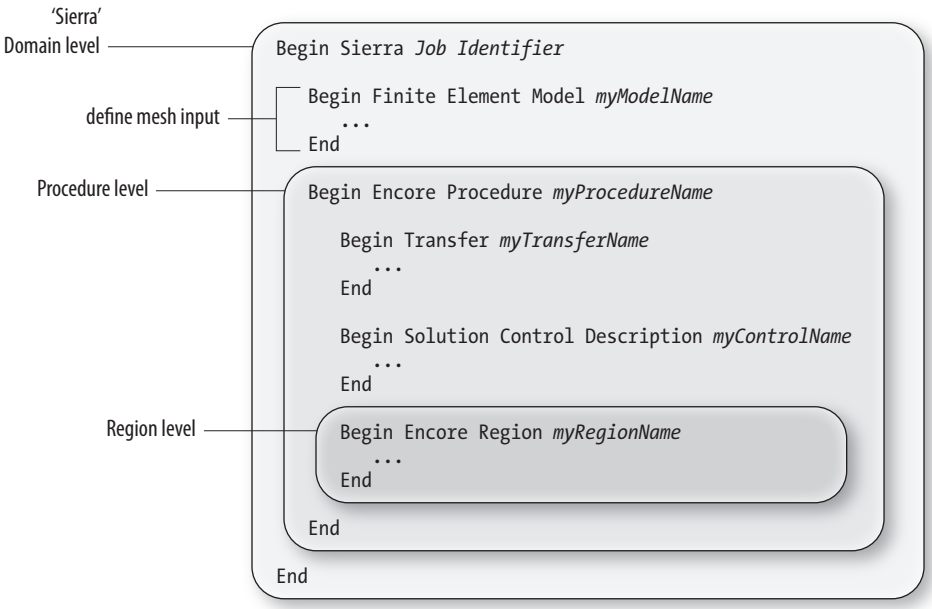
Input File Structure

Encore input is described by commands contained in an ASCII file. The structure of the input file follows a nested hierarchy. The topmost level of this hierarchy we term the *domain*, followed by the *procedure* level, and the *region* level.

Domain This top level contains one or more *procedures*. At the domain level you also find commands associated with describing the finite element mesh, fields and functions.

Procedure This 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.

Region This level is used to specify operations that are executed at each time step on a specific mesh.



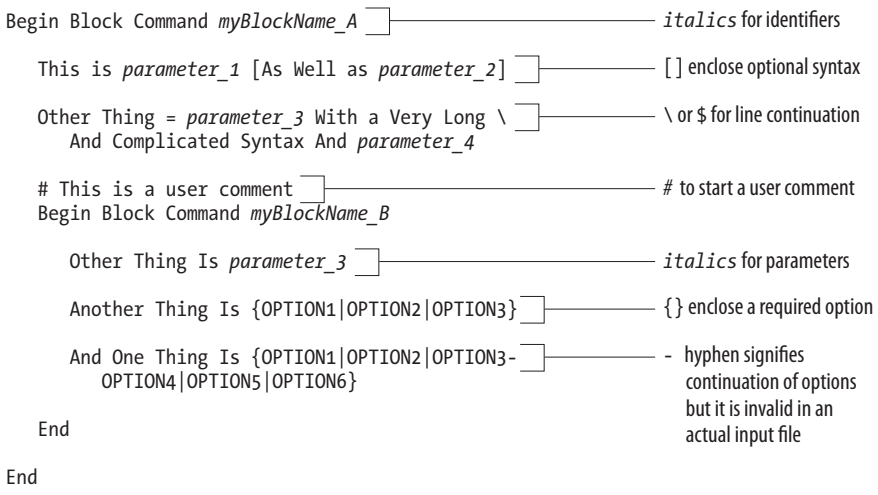
The domain, procedure, and region levels are each block commands. A block command is defined in the input file by a matching pair of `Begin` and `End` lines. A set of key words for the block command follow the `Begin` and `End`. In most cases a user-specified name is the identifier added to end of the `Begin` command line (or optionally it may be added after the `End` command).

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 terms "command block" and "block command" are interchangeable.

Conventions in this Manual

In our explanations of the syntax of input commands we use the following conventions. All input file syntax, wherever we refer to it, is given in a `monospaced font`. The beginning of all line and block commands (see the Nomenclature section below), also called keywords, have the first letter of each word capitalized. String identifiers, numeric variables, and all other kinds of user specified parameters are denoted in *`italics`*. A parameter value that is limited to one of a set of enumerated string options is denoted by enclosing the options in curly brackets {}. Sets of enumerated options that are too long to fit on one line of this manual are continued on the next line following a hyphen -. This hyphen, however,

is not actually valid syntax in a real input file. Any optional command syntax is enclosed in square brackets [].

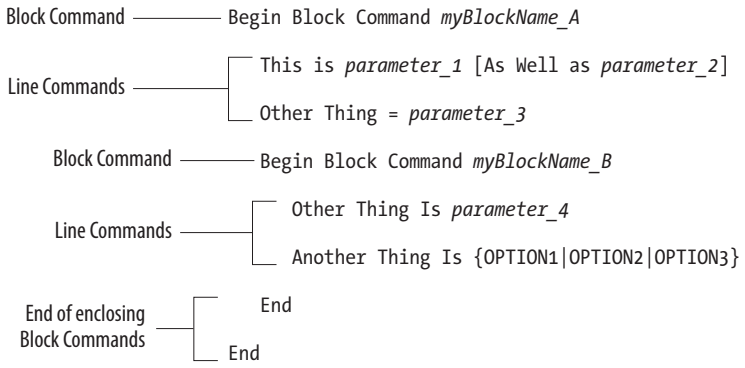


Nomenclature

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 Encore (or any Sierra Mechanics application) expects to find as it parses an input file. These are *block commands*, *line commands*, *keywords*, *names*, *parameters*, and *delimiters*.

Block and Line Commands

There are two types of commands used. 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*. Line commands are the second type of commands.



Keywords

The words which distinguish one block command, or line command, from another we term *keywords*.

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 `string` parameter in a separate line command (see below). Names are denoted in italics, *name*, as are parameters.

Parameters

There are various types of input *parameters* you may need to supply to a line command. The type of parameter(s) for a line command are always listed in a table within the synopsis of the line command. These types are:

`integer` a positive or negative integer.

`real` a real number in decimal form or exponential form. For example, `0.0001`, `.1E-3`, `10.0d-5` are all equivalent. The following are also equivalent, `1`, `1.`, or `1.0`.

`string` unquoted character string with no spaces.

`"string"` a character string enclosed by double quotation marks, which may include spaces.

`string...` one or more character strings.

`(expression)` a mathematical or boolean expression, which may include operators such as `=`, `==`, `<`, `>`, `+`, `-`, `*`, `/`, and also a limited number of standard built in functions such as `sin()`, `cos()`, `exp()`, `sqrt()`, etc.

Enumerated Parameters

Certain commands require you to specify one member from among a set of predefined identifies. This set of identifiers is called an *enumeration*, and the allowable identifies are listed within curly brackets {}, separated by the vertical line character |. If a default identifier exists, it is enclosed by <>. Specifying the default identifier has the same effect as not including the command line at all. Enumeration identifiers are case-insensitive.

Delimiters

The keywords of a line command often are required to be separated from the parameter(s) 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.

White Space

Command keywords, names, 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 recommended that you use indentation to improve readability for yourself and others.

Case Sensitivity

None of the command keywords, parameters, or delimiters, read from the input file are case sensitive. The exception to this rule are file names used for input and output, because the current operating systems on which Sierra applications run are based on UNIX or Linux operating systems, in which 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 \. 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.

Chapter 1

Functions

The concept of a *function* is important, as functions are the building blocks of your Encore input. Functions enable you to compute quantities that are of interest to you, whether you are simply postprocessing a single timestep of your grid solution, or performing a rigorous verification exercise. Encore functions correspond roughly to the idea of a mathematical function, representing field quantities that may vary in space and time. Encore functions can have the following types and their underlying representations, or ways of supplying values:

Field Function is defined piecewise on the elements, interpolated by element shape functions on the mesh or grid (using nodal or element coefficients); these are most often exist as a result of a simulation, such as temperature or displacements.

String Function is an analytic function expressed as a formula directly in the input file; these are often used to represent exact solutions for purposes of verification, but can be relatively slow to evaluate by Encore.

User Function is a subroutine or function expressed in the Fortran or C/C++ computer language; these enable you to represent functions that are either too complex or require more efficient evaluation than string functions.

The advantage of the Encore function is that it allows you to construct combinations and perform algebraic and calculus operations regardless of their underlying representations, see Figure 1.1. Some operations that can be performed using Encore functions are: taking the product or difference of two functions, averaging the function over a surface or volume, computing norms and integrals of functions, or taking the time derivatives.

Function Definition

The definition of the three basic types of functions are given inside a *`Begin/End`* function block, a distinct form for each type: `Begin Field Function`, `Begin String Function`, and `Begin User Function`. You can construct combinations of these basic types by referencing them in other blocks, such as `Begin Difference Function`.

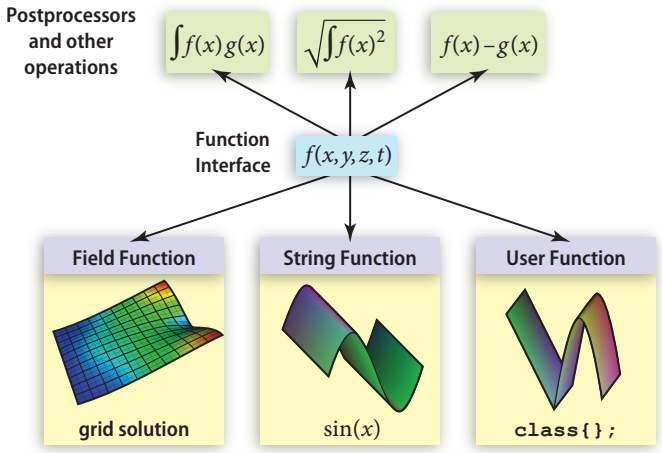


Figure 1.1. The Encore function interface hides various kinds of functions of space and time. The function interface allows you to mix and match functions in postprocessors and other operations, regardless of the underlying representation.

For calculus operations, such as those that involve derivatives and integrals, you may need to provide additional information in your function blocks. For `String` and `User` functions the spatial gradient is one example, which would be required to compute an H^1 norm. You can also specify an `Integration Order` for your function, which suggests to Encore the polynomial degree of a Gauss-Legendre quadrature rule to use during the approximation of integrals involving your function.

You can define your function with one of scalar, vector, or tensor value types. And Encore can evaluate your function in either spatial coordinates or parametric coordinates ("master element coordinates"), as appropriate with respect to various operations. String and other analytic functions are defined in spatial coordinates. Whereas, field functions or other mesh based or approximate functions, are defined in parametric coordinates. Consider the gradient of a function defined as either an analytic string function u, or as a mesh based field function u_h, the latter is computed with the standard finite element coordinate transformation and sum over shape functions, thus

$$\underset{\text{gradient of an analytic function}}{\nabla_x u(\mathbf{x}_q)} \qquad \underset{\text{gradient of mesh based field function}}{\nabla_x u_h(\mathbf{x}_q) = \textstyle\sum_i u_i \nabla_\xi \phi_i(\xi_q) J^{-1}(\mathbf{x}_q)}$$

where the summation index i is over number of nodal shape functions ϕ_i in an element, ξ_q represents the parametric, or master element, coordinates. And J^{-1} is the inverse Jacobian of the element mapping to physical coordinates. Whatever the type of function and operation, Encore automatically maps coordinates back and forth where appropriate. For example, the mapping occurs implicitly during transfers of a field function, or taking the norm of a string function.

Below we briefly discuss the three major types of functions and how they are specified in the Encore input file. Another useful input block is the *Global Function Parameters*, see below. Also useful are the blocks that combine function definitions into a new function, including the *Difference Function*, *Product Function*, and *Vector Function*.

String Functions

String functions derive their values by parsing and executing symbolic strings written directly into the input file. Their definition and intrinsic evaluation is in physical space, and when evaluating at a parametric point the string function is implicitly mapped. Here is a typical definition of string function,

```
Begin String Function sfunc
    Value Is "a+y*z+x*(1+t)"
    Gradient Is "1+t" "z" "y"
    Time Derivative Is "x"
    Integration Order Is 1
    Parameter a = 1
End
```

Symbolic strings must be enclosed in double quotes, and each vector or tensor component must each be enclosed by quotes separately. Above, notice the commands to define the value, spatial gradient, and time derivative. The `Integration Order` suggests that values of this function require at least an integration rule of polynomial degree 1. The `Parameter` command was used to define a constant referenced in the previous `Value Is` command. The `Parameter` value can be redefined later using a `Set Function Parameter` command. For example, inside a `Region` block, you could have

```
Set Function Parameter a On sfunc To 4
```

For more input files involving interpolation of string functions, their evaluation, and norms, see Example 1 and Example 2 in the Appendix.

Field Functions

Field functions are represented by a set of interpolating basis functions—typically element shape functions, defined piecewise over elements—on a mesh. Note that there are 3 different field *types*: nodal, element, and quadrature. For example, a *nodal* field function could be represented by the bilinear polynomial shape functions on a mesh of quadrilateral elements, with the coefficients equal to the values of the field at the nodes (vertices of elements). An example of an *element* field would be a piecewise constant in each quadrilateral element. And a *quadrature* field takes on multiple constant values at the element integration points, or quadrature points. You can equate an element field to a quadrature field when the number of quadrature points is one.

```
Begin Field Function my_function
    Use nodal|element|quadrature Field field_name [As Value|Gradient|Dot|Flux|Stress]
End
```

In turn, the nodal, element, and quadrature values of field functions can originate from one of three different sources: a mesh file, an application solution field, or an interpolated string function. Consider values originating from a mesh file; this is most often the case when running Encore in standalone mode. The values for the field function must first be read in from a mesh file using the `Import Field` command, inside the `Region` block (also see the section on `Region` block commands).

```
Import Field mesh_field_name As {NODAL|EDGE|FACE|ELEMENT} Field
    field_name [Of Type {SIERRA_FIELD_TYPE}]
```

The *mesh_field_name* is the name as it is stored in the Exodus file. The *field_name* is the name you want to refer back to in the `Begin Field Function` block. Usually you do not need to provide the type—it is usually determined automatically—but it can be useful to for selecting a specific component of a vector or tensor, e.g., `SYM_TENSOR_33`.

Suppose you would like to read in the nodal temperature field from a previously calculated stationary problem, and evaluate the temperature field function at $(x, y) = (0.5, 0.5)$. In addition, suppose we wish to compute the continuous L^2 norm of an analytic function. The following Encore input file does that. Evaluation at a point and various mathematical norms are further documented in the sections on the `Region` block and in the section on norms.

For an example importing a field function, see Example 3 in the Appendix.

User Function

Sometimes an analytic function is too complicated to be expressed as a String Function. This case often occurs when using the method of manufactured solutions for code verification. In such a case, you can make use of a *User Function* to write code in the C++ or Fortran language for the value, gradient, etc. The code module is then compiled and dynamically linked (as a shared object file) with Encore, or other Sierra Mechanics application.

The input syntax is

```
User Function
    Begin User Function function
    Parameter name = value
    Load From File file_name.so Using Function function_name
End
```

`Parameter name` line command allows you to specify the value of your own named parameter, which can be retrieved inside your source code for the User Function.

`file_name.so` is the name of the compiled library module (`.so` stands for shared object).

`function_name` is the symbol name of the source code function needed to register the User Function.

A user function is most easily written in C++, which derive directly from classes inside Encore. The source code below is an example of how your own C++ code would look; here it is just the basic skeleton of a user function. When you write your own class, you are overriding the virtual functions to provide evaluation of values, gradients, derivatives, etc.

```
class SomeFunc : public UserFunction
{
public:
    SomeFunc(const Fmwk::Region * reg, const Fmwk::Parameters * params)
     : UserFunction(reg, params) {}
    virtual std::vector<double> eval(...) {}
    virtual std::vector<double> gradient(...) {}
    virtual std::vector<double> dot(...) {}
};
```

For an example input file illustrating the use of a user function, and the associated C++ source, that evaluates a value, gradient and time derivative, see Example 4 in the Appendix. A second specialization of the User Function for Fortran source code is the *Fortran User Function*.

1.1 String Function

Contains the commands needed to specify a Encore function.

```
Begin String Function Function_name
   Gradient Is Exprs…
   Integration Order Is Int_order
   Parameter Name = Value
   Time Derivative Is Exprs…
   Use Function Func_name [ As Name ]
   Value Is Exprs…
End
```

Used to create a string based function.

Gradient Is

Specifies the expression to evaluate for f(x).

```
Gradient Is Exprs…
```

Exprs	(expression)…

The only acceptable variables are: x,y,z,t.

There are two different ways to specify the analytic expression desired depending on whether the function is scalar or vector valued.

If the function is scalar valued, just specify the value: GRADIENT IS "x*y+z+2"

If the function is vector valued you must specify each portion of the vector as a quoted strings seperated by spaces: GRADIENT IS "x+2" "y+z" "z*2"

Integration Order Is

Provides the Gauss integration order necessary to integrate the function exactly.

```
Integration Order Is Int_order
```

Int_order	integer

Provides the Gauss integration order necessary to integrate the function exactly.

If the function cannot be integrated exactly by a Gauss rule, then this parameter is more of a "hint" about what rule to use.

Further... this is not a hard and fast rule. Routines can ignore this parameter and use whatever integration rule they like, possibly using this parameter as a hint.

Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the function.

```
Parameter Name = Value
```

Name	string
Value	real

Time Derivative Is

Specifies the expression to evaluate for f(x).

```
Time Derivative Is Exprs…
```

Exprs	(expression)…

The only acceptable variables are: x,y,z,t.

There are two different ways to specify the analytic expression desired depending on whether the function is scalar or vector valued.

If the function is scalar valued, just specify the value: TIME DERIVATIVE IS "x*y+z+2"

If the function is vector valued you must specify each portion of the vector as a quoted strings seperated by spaces: TIME DERIVATIVE IS "x+2" "y+z" "z*2"

Use Function

Allows the function named "func_name" to be referenced inside another function by the name of "name"

```
Use Function Func_name [ As Name ]
```

Func_name	string

Value Is

Specifies the expression to evaluate for f(x).

```
Value Is Exprs…
```

Exprs	(expression)…

The only acceptable variables are: x,y,z,t.

There are two different ways to specify the analytic expression desired depending on whether the function is scalar or vector valued.

If the function is scalar valued, just specify the value: VALUE IS "x*y+z+2"

If the function is vector valued you must specify each portion of the vector as a quoted strings seperated by spaces: VALUE IS "x+2" "y+z" "z*2"

1.2 Field Function

Contains the commands needed to specify a Encore field function.

```
Begin Field Function Function_name
   Dimension Is Dimension
   Discretization Alias Is alias
   Integration Order Is Int_order
   Parameter Name = Value
   Use Option Field field_name [ As EvalFunctionTypes ]
   Use Field field_name [ As EvalFunctionTypes ]
   Use Function Func_name [ As Name ]
End
```

Used to create a field based function.

Dimension Is

Specifies the function dimension. Especially needed for Field Functions that only have values at quadrature points.

```
Dimension Is Dimension
```

Dimension	integer

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Integration Order Is

Provides the Gauss integration order necessary to integrate the function exactly.

```
Integration Order Is Int_order
```

Int_order	integer

Provides the Gauss integration order necessary to integrate the function exactly.

If the function cannot be integrated exactly by a Gauss rule, then this parameter is more of a "hint" about what rule to use.

Further... this is not a hard and fast rule. Routines can ignore this parameter and use whatever integration rule they like, possibly using this parameter as a hint.

Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the function.

```
Parameter Name = Value
```

Name	string
Value	real

Use

Specifies the field to use for the values. Optionally specify in which interface to use the data. The default is VALUE.

```
Use Option Field field_name [ As EvalFunctionTypes ]
```

field_name	string

Use Field

Specifies the field to use for the values. Optionally specify in which interface to use the data. The default is VALUE. The underlying mesh object types are determined at runtime by the FieldFunction.

```
Use Field field_name [ As EvalFunctionTypes ]
```

field_name	string

Use Function

Allows the function named "func_name" to be referenced inside another function by the name of "name"

```
Use Function Func_name [ As Name ]
```

Func_name	string

1.3 Difference Function

Contains the commands needed to specify a Encore difference function.

```
Begin Difference Function Function_name
   Difference Is Function1 - Function2
End
```

Used to create a function whose values are derived from subtracting two existing functions.

Difference Is

Specifies the functions to difference to get the values.

```
Difference Is Function1 - Function2
```

Function1	string
Function2	string

1.4 Product Function

Contains the commands needed to specify a Encore Product Function.

```
Begin Product Function Function_name
   Product Is Function1 * Function2
End
```

Used to create a function whose values are derived from multiplying two existing functions.

Product Is

Specifies the functions to multiply to get the values.

```
Product Is Function1 * Function2
```

Function1	string
Function2	string

1.5 Vector Function

Contains the commands needed to specify a Encore Vector Function.

```
Begin Vector Function Function_name
   Components Of Option Are Functions…
End
```

Used to create a vector valued function whose values are derived by specifying components of the value/gradient/etc.

Components Of

The functions to be evaluated in order to provide the components of the value/gradient/etc.

```
Components Of Option Are Functions…
```

Functions	string…

1.6 User Function

Contains the commands needed to specify a Encore user sub function.

```
Begin User Function Function_name
   Integration Order Is Int_order
   Load From File File_name Using Function Function_name
   Parameter Name = Value
   Use Function Func_name [ As Name ]
End
```

Integration Order Is

Provides the Gauss integration order necessary to integrate the function exactly.

```
Integration Order Is Int_order
```

Int_order	integer

Provides the Gauss integration order necessary to integrate the function exactly.

If the function cannot be integrated exactly by a Gauss rule, then this parameter is more of a "hint" about what rule to use.

Further... this is not a hard and fast rule. Routines can ignore this parameter and use whatever integration rule they like, possibly using this parameter as a hint.

Load From File

Specifies the path to the .so file and the registration function to call inside that .so to register the function.

```
Load From File File_name Using Function Function_name
```

File_name	string
Function_name	string

Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the function.

```
Parameter Name = Value
```

Name	string
Value	real

Use Function

Allows the function named "func_name" to be referenced inside another function by the name of "name"

```
Use Function Func_name [ As Name ]
```

Func_name	string

1.7 Fortran User Function

Contains the commands needed to specify a Encore Fortran user sub function.

```
Begin Fortran User Function Function_name
   Gradient Subroutine Is Gradient_sub
   Integration Order Is Int_order
   Load From File File_name
   Parameter Name = Value
```

```
   Use Function Func_name [ As Name ]
   Value Subroutine Is Value_sub
End
```

Gradient Subroutine Is

Specifies the name of the Fortran function to use for gradient.

```
Gradient Subroutine Is Gradient_sub
```

Gradient_sub	string

Integration Order Is

Provides the Gauss integration order necessary to integrate the function exactly.

```
Integration Order Is Int_order
```

Int_order	integer

Provides the Gauss integration order necessary to integrate the function exactly.

If the function cannot be integrated exactly by a Gauss rule, then this parameter is more of a "hint" about what rule to use.

Further... this is not a hard and fast rule. Routines can ignore this parameter and use whatever integration rule they like, possibly using this parameter as a hint.

Load From File

Specifies the path to the .so file.

```
Load From File File_name
```

File_name	string

Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the function.

```
Parameter Name = Value
```

Name	string
Value	real

Use Function

Allows the function named "func_name" to be referenced inside another function by the name of "name"

```
Use Function Func_name [ As Name ]
```

Func_name	string

Value Subroutine Is

Specifies the name of the Fortran function to use for value.

```
Value Subroutine Is Value_sub
```

Value_sub	string

1.8 Global Function Parameters

The parameters set within this block can be used by all Functions.

```
Begin Global Function Parameters Gufp
   Integration Order Is Int_order
   Parameter Name = Value
   Use Function Func_name [ As Name ]
End
```

Integration Order Is

Provides the Gauss integration order necessary to integrate the function exactly.

```
Integration Order Is Int_order
```

Int_order	integer

Provides the Gauss integration order necessary to integrate the function exactly.

If the function cannot be integrated exactly by a Gauss rule, then this parameter is more of a "hint" about what rule to use.

Further... this is not a hard and fast rule. Routines can ignore this parameter and use whatever integration rule they like, possibly using this parameter as a hint.

Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the function.

```
Parameter Name = Value
```

Name	string
Value	real

Use Function

Allows the function named "func_name" to be referenced inside another function by the name of "name"

```
Use Function Func_name [ As Name ]
```

Func_name	string

Chapter 2

Postprocessors and Norms

The concept of an Encore *Postprocessor* is that of a calculation that is executed at a specific point in the simulation, or after the simulation. Thus the name Postprocessor is a bit of a misnomer, because it can run at almost any time. Therefore, an Encore "post"-processor could—in some cases—act as a "pre"-processor. Encore postprocessors can be run at the following points in time, among others,

- after initial conditions,
- after a non-linear step,
- after every timestep,
- at a certain simulation time.

Postprocessor Definition

The input to most postprocessors are Encore *Functions*. And the output of a postprocessor can be values of simulation fields (a nodal field, element field, etc.), or a single global value written to a formatted table in an output text file. The output can take the form of scalars, vectors, and tensor values. Postprocessors run in parallel, if the Sierra application that contains it is run in parallel.

Many postprocessors are built-in to Encore. Commonly used examples are the `Min/Max Postprocessor` and the `Average Value Postprocessor`. Another is the `Evaluate Function Postprocessor` which is used to evaluate an Encore Function at a point, see Example 1, Example 3, and Example 4 in the Appendix. More complex built-in postprocessors include the `Recover Function Postprocessor`, which can output a nodal field containing the smoothened values of the derivative of a nodal field or the values of an element field. Another is the `Surface Normal Processor`, which can compute the surface flux of a given `Function`. Finally, an example of a built-in postprocessor that combines the output of two other postprocessors to result in a single output, is the `Difference Postprocessor`.

In addition to the built-in Postprocessors, you can extend Encore by writing a `User Postprocessor`, which is similar to writing a User Function, see also section 1. An example

input file and the C++ source code for a User Postprocessor is given in Example 7 in the Appendix.

There are two important utilities related to postprocessors. These utilities have input syntax blocks of their own. The first is the `Postprocessor Group`, which allows you to execute multiple postprocessors at the same time. The second is the `Postprocessor Output Control`, which specifies the file name and format for tabular output.

Norms

A special category of Encore's built-in postprocessors are *Norms*. Norms equip a vector space, or a function space, with a way to measure the "length" of a member of the space, the "distance" between two members, and the "inner product" between two members. Norms all obey certain basic rules, such as the Hölder and Minkowski inequalities. Encore has built-in commands for common *discrete (sum)* and *continuous (integral) norms*.

In Encore input syntax the discrete, or finite sum, norms are called *nodal norms*. They are called nodal norms because the way in which they are computed on a field is by summing the field coefficients associated with each node on the mesh. The following nodal and continuous norms are built-in: the L_1, L_2, L_∞ norms, and the H_1-seminorm (which is denoted as H_1). In addition, *relative* norms for each of those are available.

You have a choice of two styles of writing the input syntax for the norms. You can write the norm either as part of a `Norm Postprocessor` block command at the `Domain` level, or as a single line command in an enclosing `Region` block. The former block command style is more clean if you want to execute multiple norms and/or postprocessors together. For examples of both these styles, see Example 5 for the block command style, and Example 6 for the line command style, in the Appendix.

2.1 Quadratic Difference Postprocessor

Compute quadratic difference between data values and Postprocessor output.

```
Begin Quadratic Difference Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Data Values u…
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Postprocessor Pp_name
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Data Values

Values to use in quadratic difference with PP values.

```
Data Values u…
```

u	real…

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Postprocessor

Specifies the postprocessor to evaluate.

```
Use Postprocessor Pp_name
```

Pp_name	string

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.2 Equivalent Cure Postprocessor

Contains the commands needed to specify a postprocessor that integrates the equivalent cure quantity, an integral over time at each node.

```
Begin Equivalent Cure Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Store In field_name
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Store In

Specify the name of the field in which you want to store the result.

```
Store In field_name
```

field_name	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.3 Tabular Function Output Postprocessor

Write tabular output of functions at nodal values.

```
Begin Tabular Function Output Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
   Write To File Filename
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

Write To File

Specifies the name of a file to write the tabular results to.

```
Write To File Filename
```

Filename	string

2.4 Postprocessor Group

Gives a name to the enclosed group of postprocessors.

```
Begin Postprocessor Group Group_name
   Compute Difference NormTypes Of f1 f2 [ Store In field_name ]
   Compute Norm NormTypes Of Function_name [ Store In field_name ]
   Create Edge Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
   Create Element Field Field_name Of Type Option And Dimension Dimension [ On Part_list…
      ]
   Create Face Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
   Create Nodal Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
   Create Quadrature Field Field_name Of Type Option And Order Intg_order
   Evaluate Flux Of Function_name At x [ y z t ]
   Evaluate Function Function_name At x [ y z t ]
   Evaluate Gradient Of Function_name At x [ y z t ]
   Evaluate Postprocessor Postprocessor_name
```

```
   Evaluate Time Derivative Of Function_name At x [ y z t ]
   Execute On Output
   Execute Postprocessor Group Postprocessor_name
   Import Field Mesh_field_name As Option1 Field Field_name [ Of Type Option2 ]
   Indicatemarkadapt Using Indicator Marker
   Interpolate Function Option1 Of Function_name Into Option2 Field Field_name [ On Op-
      tion3 Part_list… ]
   Markadapt Using Marker
   Output Indicator Global Value Of Indicator_name
   Output Number Of Option
   Output Postprocessor History Of Pp_name
   Output Preferred Integration Order Of Function_name
   Print Values Of Option Field Field_name
   Process Initial Condition
   Set Function Parameter Param_name On Function_name To Value
   Verify Hanging Node Constraints
End
```

Compute Difference

Computes the error between the two functions using the specified norm.
In the case of relative norms it does norm(f1-f2) / norm(f1) .

```
Compute Difference NormTypes Of f1 f2 [ Store In field_name ]
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}
f1	string
f2	string

Compute Norm

Computes the norm of the function using the specified norm.

```
Compute Norm NormTypes Of Function_name [ Store In field_name ]
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}
Function_name	string

Create Edge Field

Creates an edge Field named field_name on the region.

```
Create Edge Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Element Field

Creates an Element Field name field_name on the region.

```
Create Element Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Face Field

Creates a face Field named field_name on the region.

```
Create Face Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Nodal Field

Creates a Nodal Field name field_name on the region.

```
Create Nodal Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Quadrature Field

Creates an Element Field named field_name sized correctly to receive a quadrature transfer of the order specified.

```
Create Quadrature Field Field_name Of Type Option And Order Intg_order
```

Field_name	string
Intg_order	integer

Evaluate Flux Of

Evaluates the gradient of the function at the given point.

```
Evaluate Flux Of Function_name At x [ y z t ]
```

Function_name	string
x	real

Evaluate Function

Evaluates the function at the given point.

```
Evaluate Function Function_name At x [ y z t ]
```

Function_name	string
x	real

Evaluate Gradient Of

Evaluates the gradient of the function at the given point.

```
Evaluate Gradient Of Function_name At x [ y z t ]
```

Function_name	string
x	real

Evaluate Postprocessor

Evaluates the specified processor at the current time.

```
Evaluate Postprocessor Postprocessor_name
```

Postprocessor_name	string

Evaluate Time Derivative Of

Evaluates the time derivative of the function at the given point.

```
Evaluate Time Derivative Of Function_name At x [ y z t ]
```

Function_name	string
x	real

Execute On Output

Causes this Postprocessor Group to be executed just before output happens. Useful if you are only interested in seeing the PP output in your Results Output and not at any other times.

```
Execute On Output
```

Execute Postprocessor Group

Executes all of the postprocessors in the specified group.

```
Execute Postprocessor Group Postprocessor_name
```

Postprocessor_name	string

Import Field

Imports a field from the current mesh and copies it to a SIERRA field. The field must be nodal or element. The optional parameter is the time at which to read the field.

```
Import Field Mesh_field_name As Option1 Field Field_name [ Of Type Option2 ]
```

Mesh_field_name	string
Field_name	string

Indicatemarkadapt Using

Postprocessor that simulates the solution control IndicateMarkAdapt line command. When this Postprocessor executes the region will be immediately adapted once. To get more complicated execution (for instance more levels of refinement) you must use Solution Control.

```
Indicatemarkadapt Using Indicator Marker
```

Indicator	string
Marker	string

Interpolate Function

Evaluates the specified function at the nodal positions and stores the result in field_name. Optionally, a list of surfaces or volumes can be listed that restrict the interpolation.

```
Interpolate Function Option1 Of Function_name Into Option2 Field Field_name [ On Option3
   Part_list… ]
```

Function_name	string
Field_name	string

Markadapt Using

Postprocessor that simulates the solution control MarkAdapt line command. When this Postprocessor executes the region will be immediately adapted once. To get more complicated execution (for instance more levels of refinement) you must use Solution Control.

```
Markadapt Using Marker
```

Marker	string

Output Indicator Global Value Of

Outputs the global value of an indicator. This is usually the summation of each element's error contribution.

```
Output Indicator Global Value Of Indicator_name
```

Indicator_name	string

Output Number Of

A postprocessor that just outputs the current number of active nodes or elements.

```
Output Number Of Option
```

Output Postprocessor History Of

Outputs the cumulative PP history values.

```
Output Postprocessor History Of Pp_name
```

Pp_name	string

Output Preferred Integration Order Of

Outputs the preferred integration order of the function.

```
Output Preferred Integration Order Of Function_name
```

Function_name	string

Print Values Of

Prints the values of a field to the screen.

```
Print Values Of Option Field Field_name
```

Field_name	string

Process Initial Condition

This causes the first time step to be at time 0 when importing a solution... allowing post-processing and transfers of the initial condition.

It should probably only be used when running Encore as a standalone application, as it might throw other applications off.

```
Process Initial Condition
```

Set Function Parameter

Sets the value of "param" on function "function_name" to the specified value.

```
Set Function Parameter Param_name On Function_name To Value
```

Param_name	string
Function_name	string
Value	real

Verify Hanging Node Constraints

Verify hanging node constraints for nodal fields.

```
Verify Hanging Node Constraints
```

2.5 User Postprocessor

Contains the commands needed to specify a Encore user sub postprocessor.

```
Begin User Postprocessor Function_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Load From File File_name Using Function Function_name
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Parameter Name = Value
   Store In field_name
   String Parameter Name = Value
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Vector Parameter Name = Values…
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Load From File

Specifies the path to the .so file and the registration function to call inside that .so to register the postprocessor.

```
Load From File File_name Using Function Function_name
```

File_name	string
Function_name	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the PP.

```
Parameter Name = Value
```

Name	string
Value	real

Store In

Specify the name of the field in which you want to store the result.

```
Store In field_name
```

field_name	string

String Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the PP.

```
String Parameter Name = Value
```

Name	string
Value	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Vector Parameter

Creates a parameter with the name and values specified. This parameter name can then be referenced by the PP.

```
Vector Parameter Name = Values…
```

Name	string
Values	real…

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.6 Sierra Kl Solver Error Indicator

Compute error indicators for solutions of KL eigenproblems.

```
Begin Sierra Kl Solver Error Indicator Name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Covariance Length Scale = Scale…
   Covariance Type = Option
   Discretization Alias Is alias
   Eigenvalue Name = lambda_name
   Eigenvector Name = phi_name
   Integration Order = Intg_order
   Mesh Object Type = Option
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Store In field_name
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Internal Sierra Kl Data
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Covariance Length Scale

Specifies the length scale to use in the covariance kernel function. If multiple values are specified, they are interpreted as length scales in each coordinate direction

```
Covariance Length Scale = Scale…
```

Scale	real…

Covariance Type

Specifies the type of covariance kernel function.

```
Covariance Type = Option
```

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Eigenvalue Name

Specifies the name of the global variable containing the KL eigenvalues.

```
Eigenvalue Name = lambda_name
```

lambda_name	string

Eigenvector Name

Specifies the name of the vector-valued field containing the KL eigenvectors.

```
Eigenvector Name = phi_name
```

phi_name	string

Integration Order

Specifies the order of integration rule (must be positive).

```
Integration Order = Intg_order
```

Intg_order	integer

Mesh Object Type

Specifies the element type for the field.

```
Mesh Object Type = Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Store In

Specify a field in which to store the errors in the eigenfunctions.

```
Store In field_name
```

field_name	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Internal Sierra Kl Data

Specifies to use Sierra internal fields and global variables rather than from input file.

```
Use Internal Sierra Kl Data
```

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.7 Realize Pc Random Field

Compute realizations of random fields using PCs.

```
Begin Realize Pc Random Field Name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Mesh Object Type = Option
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Pc Polynomial Degree = poly_degree
   Random Field Basis Name = Rf_basis_name
   Random Field Name = Rf_name
   Random Variable Values = values…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Mesh Object Type

Specifies the size of the vector of standard normal random variable values.

```
Mesh Object Type = Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Pc Polynomial Degree

Specifies the global degree of the PC expansion.

```
Pc Polynomial Degree = poly_degree
```

poly_degree	integer

Random Field Basis Name

Specifies the name of the vector-valued field containing the random field basis vectors.

```
Random Field Basis Name = Rf_basis_name
```

Rf_basis_name	string

Random Field Name

Specifies the name of the field containing the random field realization.

```
Random Field Name = Rf_name
```

Rf_name	string

Random Variable Values

Specifies the vector of standard normal random variable values. These are used to evaluate the coefficient functions of the random field basis.

```
Random Variable Values = values…
```

values	real…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.8 Summation Postprocessor

Computes the sum of the nodal values of the function for all nodes in the volume / on the surface.

```
Begin Summation Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Loop Type Option
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Loop Type

Specify the type of object to loop over, e.g. node, edge, face, element, etc.x

```
Loop Type Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.9 Postprocessor Output Control

Contains the command needed to specify parameters dealing with output from post-processors

```
Begin Postprocessor Output Control Name
   Comment Character Is Character
   Enable Small Output Rounding To Zero
   Floating Point Format Is Option
   Floating Point Precision Is Precision
   Output Time
   Output To Console
   Output To Log File
   Separator Character Is Exprs…
   Write To File Filename
End
```

Comment Character Is

This will cause the comment character specified to appear before the title and other header lines in the printed table.

```
Comment Character Is Character
```

Character	string

Enable Small Output Rounding To Zero

Turn on rounding of small output numbers to zero. This feature is intended for regression testing only, and only applies to fixed format output.

```
Enable Small Output Rounding To Zero
```

Floating Point Format Is

Specifies the floating point format.

```
Floating Point Format Is Option
```

Floating Point Precision Is

Specifies the number of digits to preserve when outputing numbers.

```
Floating Point Precision Is Precision
```

Precision	integer

Output Time

Forces a Time column to be output.

```
Output Time
```

Output To Console

If this line exists then output will be done on the console.

```
Output To Console
```

Output To Log File

If this line exists then output will go out to the Sierra log file.

```
Output To Log File
```

Separator Character Is

This will cause the separator character specified to be used between output values of each postprocessor. The default is a comma.

```
Separator Character Is Exprs…
```

Exprs	(expression)…

Write To File

Specifies the name of a file to write the results to.

```
Write To File Filename
```

Filename	string

2.10 Evaluate Function Postprocessor

Contains the commands needed to specify a postprocessor for evaluating a function at a point. To enable logging of search add `-O "-encorelog search -pout"` to the sierra script. For serial runs the `"-pout"` can be dropped.

```
Begin Evaluate Function Postprocessor Postprocessor_name
   At Nearest Node
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Evaluate EvalFunctionTypes
   Location x [ y z ]
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Parametric Search Tolerance Tol
   Surfaces Surface_list…
   Surfaces All
   Time t
   Track Point
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
```

```
   Volumes Volume_list…
   Volumes All
End
```

At Nearest Node

Causes the evaluation to happen at the nearest node to the location given.

```
At Nearest Node
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Evaluate

Evaluates the function in a specific way. If this line command is not present, the default will be to evaluate the Value.

```
Evaluate EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Location

Evaluates the function at the given point.

```
Location x [ y z ]
```

x	real

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Parametric Search Tolerance

Set a parametric tolerance used to allow for points to be interpolated that lie just outside of an element.

```
Parametric Search Tolerance Tol
```

Tol	real

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Time

Evaluates the function at the given time. If not specified then the current simulation time is used instead. Note that this has no effect on field functions.

```
Time t
```

t	real

Track Point

Causes the actual place where the function gets evaluated to move with mesh motion.

```
Track Point
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.11 Least Squares Difference Postprocessor

Compute least squares fit between data points and Function.

```
Begin Least Squares Difference Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Data Value u At Point x y z
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
```

```
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Data Value

duh

```
Data Value u At Point x y z
```

u	real
x	real
y	real
z	real

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.12 Ratio Postprocessor

Take ratio of two postprocessors.

```
Begin Ratio Postprocessor Postprocessor_name
   Ratio Is Postprocessor1 / Postprocessor2
End
```

Ratio Is

Specify the Postprocessors to use in the ratio.

```
Ratio Is Postprocessor1 / Postprocessor2
```

Postprocessor1	string
Postprocessor2	string

2.13 Realize Random Field

Compute realizations of random fields.

```
Begin Realize Random Field Name
   Coefficient Name = Rf_coeff_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Constant Mean Value = Value
   Constant Variance Scale Factor = Value
   Discretization Alias Is alias
   Mesh Object Type = Option
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Random Field Basis Name = Rf_basis_name
   Random Field Name = Rf_name
   Random Variable Values = values…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Internal Sierra Kl Data
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Coefficient Name

Specifies the name of the coefficient for use in scaling the random field basis vectors (for use in K-L expansions only).

```
Coefficient Name = Rf_coeff_name
```

Rf_coeff_name	string

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Constant Mean Value

Specifies the (constant) mean value of the random field.

```
Constant Mean Value = Value
```

Value	real

Constant Variance Scale Factor

Specifies a constant variance scale factor of the random field.

```
Constant Variance Scale Factor = Value
```

Value	real

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Mesh Object Type

Specifies the size of the vector of standard normal random variable values.

```
Mesh Object Type = Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Random Field Basis Name

Specifies the name of the vector-valued field containing the random field basis vectors.

```
Random Field Basis Name = Rf_basis_name
```

Rf_basis_name	string

Random Field Name

Specifies the name of the field containing the random field realization.

```
Random Field Name = Rf_name
```

Rf_name	string

Random Variable Values

Specifies the vector of standard normal random variable values. These are used to evaluate the coefficient functions of the random field basis.

```
Random Variable Values = values…
```

values	real…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Internal Sierra Kl Data

Specifies to use Sierra internal fields and global variables rather than from input file.

```
Use Internal Sierra Kl Data
```

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.14 Adjoint Test Postprocessor

Test adjoint solver just copies rhs Field to solution Field.

```
Begin Adjoint Test Postprocessor Postprocessor_name
   Compute Nodal Rhs Field Rhs_name
   Compute Nodal Sensitivity Field Field_name
   Compute Nodal Solution Field Solution_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Rhs Postprocessor Rhs_pp
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Rhs Field

Specify the name of the nodal right hand side Field to the adjoint solver.

```
Compute Nodal Rhs Field Rhs_name
```

Rhs_name	string

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Nodal Solution Field

Specify the name of the nodal solution Field to the adjoint solver.

```
Compute Nodal Solution Field Solution_name
```

Solution_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Rhs Postprocessor

Specify the name of the rhs postprocessor. Must be derived from SensitivePostprocessor.

```
Rhs Postprocessor Rhs_pp
```

Rhs_pp	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.15 Average Value Postprocessor

Contains the commands needed to specify a postprocessor for the average value of a function over the domain.

```
Begin Average Value Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
```

```
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.16 Norm Postprocessor

Compute the norm of a function.

```
Begin Norm Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Norm NormTypes
   Compute Norms NormTypes…
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Store In field_name
   Subtract Function Function_name
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Norm

Computes the norm of the function using the specified norm.

```
Compute Norm NormTypes
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}

Compute Norms

Computes the norm of the function using the specified norm.

```
Compute Norms NormTypes…
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Store In

Specify the name of the field in which you want to store the result.

```
Store In field_name
```

field_name	string

Subtract Function

Specifies the function to subtract from the other function for differences.

```
Subtract Function Function_name
```

Function_name	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.17 Mesh Size From Marker

Converts a marker field (integer,element) into a mesh size field (double,element).

```
Begin Mesh Size From Marker Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Store In MeshSizeField
   Surfaces Surface_list…
   Surfaces All
   Use Element Field Marker_field
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Store In

Specify the name of the element field that will contain the mesh size values.

```
Store In MeshSizeField
```

MeshSizeField	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Element Field

Specify the name of the integer element field that contains the marker values.

```
Use Element Field Marker_field
```

Marker_field	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.18 Recover Function Postprocessor

Contains the commands needed to specify a postprocessor for recover a function value, gradient, etc. into a Field.

```
Begin Recover Function Postprocessor Postprocessor_name
   Averaging Is least squares|simple average
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Polynomial Degree Degree
   Recover EvalFunctionTypes
   Store In recover_field_list…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Averaging Is

Specifies what kind of averaging to use.

```
Averaging Is least squares|simple average
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Polynomial Degree

Specify the polynomial degree to use for the recovery.

```
Polynomial Degree Degree
```

Degree	integer

Recover

Recovers the function in a specific way. If this line command is not present, the default will be to recover the Value.

```
Recover EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Store In

Specify the list of Fields in which to store the recovered values.

```
Store In recover_field_list…
```

recover_field_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.19 Percent Threshold Postprocessor

Contains the commands needed to specify a postprocessor for computing the percent of function values within upper and lower thresholds. Values are evaluated at a given mesh object type.

```
Begin Percent Threshold Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Lower Threshold = x
   Mesh Object Type = Option
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Surfaces Surface_list…
   Surfaces All
   Upper Threshold = x
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Lower Threshold

Restrict values counted to those greater than or equal to this value.

```
Lower Threshold = x
```

x	real

Mesh Object Type

Specifies the type of mesh object to loop over.

```
Mesh Object Type = Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Upper Threshold

Restrict values counted to those less than or equal to this value.

```
Upper Threshold = x
```

x	real

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.20 Mesh Size From Error

Converts a (double,element) error field into a mesh size field (double,element).

```
Begin Mesh Size From Error Postprocessor_name
   Asymptotic Error Rate Is rate
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Error Reduction Factor Is factor
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Store In Mesh_size_field
   Surfaces Surface_list…
   Surfaces All
   Use Element Field Error_field
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Asymptotic Error Rate Is

Specify the asymptotic error rate of the element error indicator.

```
Asymptotic Error Rate Is rate
```

rate	real

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Error Reduction Factor Is

Specify the desired reduction factor for the initial global error.

```
Error Reduction Factor Is factor
```

factor	real

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Store In

Specify the name of the element field that will contain the mesh size values.

```
Store In Mesh_size_field
```

Mesh_size_field	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Element Field

Specify the name of the element field that contains the error values.

```
Use Element Field Error_field
```

Error_field	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.21 Surface Normal Postprocessor

Contains the commands needed to specify a postprocessor for the surface flux of a function over a list of sidesets.

```
Begin Surface Normal Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Evaluate EvalFunctionTypes
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Evaluate

Evaluates the function in a specific way. If this line command is not present, the default will be to evaluate the Value.

```
Evaluate EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.22 Sierra Kl Solver

Compute eigenvalues/eigenvectors for a discrete KL solve given a covariance function.

```
Begin Sierra Kl Solver Name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Covariance Length Scale = Scale…
   Covariance Type = Option
   Discretization Alias Is alias
   Enable Matrix Free
   Integration Order = Intg_order
   Maximum Number Of Eigenvalues = Max_evs
   Mesh Object Type = Option
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Store Eigenvalues In Eval_name
   Store Eigenvectors In Evec_name
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Covariance Length Scale

Specifies the length scale to use in the covariance kernel function. If multiple values are specified, they are interpreted as length scales in each coordinate direction

```
Covariance Length Scale = Scale…
```

Scale	real…

Covariance Type

Specifies the type of covariance kernel function.

```
Covariance Type = Option
```

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Enable Matrix Free

Enable matrix free solver.

```
Enable Matrix Free
```

Integration Order

Specifies the order of integration rule (must be positive).

```
Integration Order = Intg_order
```

Intg_order	integer

Maximum Number Of Eigenvalues

Specifies the maximum number of eigenvalues to compute.

```
Maximum Number Of Eigenvalues = Max_evs
```

Max_evs	integer

Mesh Object Type

Specifies the size of the vector of standard normal random variable values.

```
Mesh Object Type = Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Store Eigenvalues In

Specifies the name of the global variable in which to store the eigenvalues.

```
Store Eigenvalues In Eval_name
```

Eval_name	string

Store Eigenvectors In

Specifies the name of the field in which to store the eigenvectors.

```
Store Eigenvectors In Evec_name
```

Evec_name	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.23 Min Max Postprocessor

Computes the maximum and minimum of the nodal values of the function for all nodes in the volume / on the surface.

```
Begin Min Max Postprocessor Postprocessor_name
   Compute Option
   Compute Nodal Sensitivity Field Field_name
```

```
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Loop Type Option
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Output Global Id
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute

Specify the type of output: min, max or both.

```
Compute Option
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Loop Type

Specify the type of object to loop over, e.g. node, edge, face, element, etc.

```
Loop Type Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Output Global Id

Print the global IDs of the nodes/elements where the min/max occurs.

```
Output Global Id
```

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.24 Integral Least Squares Difference Postprocessor

Compute the integral least squares fit between data points and Function.

```
Begin Integral Least Squares Difference Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Surfaces Surface_list…
   Surfaces All
   Use Data Function Data_function_name
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Data Function

Specifies the data function to subtract from the use function.

```
Use Data Function Data_function_name
```

Data_function_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.25 Integrate Function Postprocessor

Contains the commands needed to specify a postprocessor that integrates the given function over the provided meshparts.

```
Begin Integrate Function Postprocessor Postprocessor_name
   Compute Nodal Sensitivity Field Field_name
   Compute Sensitivity Field Field_name
   Discretization Alias Is alias
```

```
   Nodes Nodeset_list…
   Omit Volumes Volume_list…
   Surfaces Surface_list…
   Surfaces All
   Use Function Function_list
   Use Functions Function_list…
   Use Weight Function Weight_Function
   Volumes Volume_list…
   Volumes All
End
```

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

2.26 Difference Postprocessor

Take difference of two postprocessors.

```
Begin Difference Postprocessor Postprocessor_name
   Difference Is Postprocessor1 - Postprocessor2
End
```

Difference Is

Specify the Postprocessors to difference to get the values.

```
Difference Is Postprocessor1 - Postprocessor2
```

Postprocessor1	string
Postprocessor2	string

Asymptotic Error Rate Is

Specify the asymptotic error rate of the element error indicator.

```
Asymptotic Error Rate Is rate
```

rate	real

At Nearest Node

Causes the evaluation to happen at the nearest node to the location given.

```
At Nearest Node
```

Averaging Is

Specifies what kind of averaging to use.

```
Averaging Is least squares|simple average
```

Coefficient Name

Specifies the name of the coefficient for use in scaling the random field basis vectors (for use in K-L expansions only).

```
Coefficient Name = Rf_coeff_name
```

Rf_coeff_name	string

Comment Character Is

This will cause the comment character specified to appear before the title and other header lines in the printed table.

```
Comment Character Is Character
```

Character	string

Compute

Specify the type of output: min, max or both.

```
Compute Option
```

Compute Nodal Rhs Field

Specify the name of the nodal right hand side Field to the adjoint solver.

```
Compute Nodal Rhs Field Rhs_name
```

Rhs_name	string

Compute Nodal Sensitivity Field

Specify a nodal sensitivity field for FieldFunction sensitivities.

```
Compute Nodal Sensitivity Field Field_name
```

Field_name	string

Compute Nodal Solution Field

Specify the name of the nodal solution Field to the adjoint solver.

```
Compute Nodal Solution Field Solution_name
```

Solution_name	string

Compute Norm

Computes the norm of the function using the specified norm.

```
Compute Norm NormTypes
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}

Compute Norms

Computes the norm of the function using the specified norm.

```
Compute Norms NormTypes…
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}

Compute Sensitivity Field

Specify a sensitivity field for FieldFunction sensitivities.

```
Compute Sensitivity Field Field_name
```

Field_name	string

Constant Mean Value

Specifies the (constant) mean value of the random field.

```
Constant Mean Value = Value
```

Value	real

Constant Variance Scale Factor

Specifies a constant variance scale factor of the random field.

```
Constant Variance Scale Factor = Value
```

Value	real

Covariance Length Scale

Specifies the length scale to use in the covariance kernel function. If multiple values are specified, they are interpreted as length scales in each coordinate direction

```
Covariance Length Scale = Scale…
```

Scale	real…

Covariance Length Scale

Specifies the length scale to use in the covariance kernel function. If multiple values are specified, they are interpreted as length scales in each coordinate direction

```
Covariance Length Scale = Scale…
```

Scale	real…

Covariance Type

Specifies the type of covariance kernel function.

```
Covariance Type = Option
```

Covariance Type

Specifies the type of covariance kernel function.

```
Covariance Type = Option
```

Data Value

duh

```
Data Value u At Point x y z
```

u	real
x	real
y	real
z	real

Data Values

Values to use in quadratic difference with PP values.

```
Data Values u…
```

u	real…

Difference Is

Specify the Postprocessors to difference to get the values.

```
Difference Is Postprocessor1 - Postprocessor2
```

Postprocessor1	string
Postprocessor2	string

Discretization Alias Is

Specifies the discretization alias to use for the interpolation of the field values. Examples include Q1 (linear), Q2 (quadratic), P0 (constant), etc.

```
Discretization Alias Is alias
```

alias	string

Eigenvalue Name

Specifies the name of the global variable containing the KL eigenvalues.

```
Eigenvalue Name = lambda_name
```

lambda_name	string

Eigenvector Name

Specifies the name of the vector-valued field containing the KL eigenvectors.

```
Eigenvector Name = phi_name
```

phi_name	string

Enable Matrix Free

Enable matrix free solver.

```
Enable Matrix Free
```

Enable Small Output Rounding To Zero

Turn on rounding of small output numbers to zero. This feature is intended for regression testing only, and only applies to fixed format output.

```
Enable Small Output Rounding To Zero
```

Error Reduction Factor Is

Specify the desired reduction factor for the initial global error.

```
Error Reduction Factor Is factor
```

factor	real

Evaluate

Evaluates the function in a specific way. If this line command is not present, the default will be to evaluate the Value.

```
Evaluate EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Evaluate

Evaluates the function in a specific way. If this line command is not present, the default will be to evaluate the Value.

```
Evaluate EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Evaluate Postprocessor

Evaluates the specified processor at the current time.

```
Evaluate Postprocessor Postprocessor_name
```

Postprocessor_name	string

Execute On Output

Causes this Postprocessor Group to be executed just before output happens. Useful if you are only interested in seeing the PP output in your Results Output and not at any other times.

```
Execute On Output
```

Execute Postprocessor Group

Executes all of the postprocessors in the specified group.

```
Execute Postprocessor Group Postprocessor_name
```

Postprocessor_name	string

Floating Point Format Is

Specifies the floating point format.

```
Floating Point Format Is Option
```

Floating Point Precision Is

Specifies the number of digits to preserve when outputing numbers.

```
Floating Point Precision Is Precision
```

Precision	integer

Indicatemarkadapt Using

Postprocessor that simulates the solution control IndicateMarkAdapt line command. When this Postprocessor executes the region will be immediately adapted once. To get more complicated execution (for instance more levels of refinement) you must use Solution Control.

```
Indicatemarkadapt Using Indicator Marker
```

Indicator	string
Marker	string

Initial Markadapt Using

Causes the marker listed to be used for num iterations and an adaptive step done on the Region before the first execution of the Region. This also precedes initialization of the Region, including setting initial conditions.

```
Initial Markadapt Using Marker For Num Iterations
```

Marker	string
Num	integer

Integration Order

Specifies the order of integration rule (must be positive).

```
Integration Order = Intg_order
```

Intg_order	integer

Integration Order

Specifies the order of integration rule (must be positive).

```
Integration Order = Intg_order
```

Intg_order	integer

Load From File

Specifies the path to the .so file and the registration function to call inside that .so to register the postprocessor.

```
Load From File File_name Using Function Function_name
```

File_name	string
Function_name	string

Location

Evaluates the function at the given point.

```
Location x [ y z ]
```

x	real

Loop Type

Specify the type of object to loop over, e.g. node, edge, face, element, etc.x

```
Loop Type Option
```

Loop Type

Specify the type of object to loop over, e.g. node, edge, face, element, etc.

```
Loop Type Option
```

Lower Threshold

Restrict values counted to those greater than or equal to this value.

```
Lower Threshold = x
```

x	real

Markadapt Using

Postprocessor that simulates the solution control MarkAdapt line command. When this Postprocessor executes the region will be immediately adapted once. To get more complicated execution (for instance more levels of refinement) you must use Solution Control.

```
Markadapt Using Marker
```

Marker	string

Maximum Number Of Eigenvalues

Specifies the maximum number of eigenvalues to compute.

```
Maximum Number Of Eigenvalues = Max_evs
```

Max_evs	integer

Mesh Object Type

Specifies the type of mesh object to loop over.

```
Mesh Object Type = Option
```

Mesh Object Type

Specifies the element type for the field.

```
Mesh Object Type = Option
```

Mesh Object Type

Specifies the size of the vector of standard normal random variable values.

```
Mesh Object Type = Option
```

Mesh Object Type

Specifies the size of the vector of standard normal random variable values.

```
Mesh Object Type = Option
```

Mesh Object Type

Specifies the size of the vector of standard normal random variable values.

```
Mesh Object Type = Option
```

Nodes

Specify all of the nodesets you wish to include in the postprocessor.

```
Nodes Nodeset_list…
```

Nodeset_list	string…

Omit Volumes

Specify all of the volumes you wish to omit from the postprocessor.

```
Omit Volumes Volume_list…
```

Volume_list	string…

Output Global Id

Print the global IDs of the nodes/elements where the min/max occurs.

```
Output Global Id
```

Output Indicator Global Value Of

Outputs the global value of an indicator. This is usually the summation of each element's error contribution.

```
Output Indicator Global Value Of Indicator_name
```

Indicator_name	string

Output Number Of

A postprocessor that just outputs the current number of active nodes or elements.

```
Output Number Of Option
```

Output Postprocessor History Of

Outputs the cumulative PP history values.

```
Output Postprocessor History Of Pp_name
```

Pp_name	string

Output Preferred Integration Order Of

Outputs the preferred integration order of the function.

```
Output Preferred Integration Order Of Function_name
```

Function_name	string

Output Time

Forces a Time column to be output.

```
Output Time
```

Output To Console

If this line exists then output will be done on the console.

```
Output To Console
```

Output To Log File

If this line exists then output will go out to the Sierra log file.

```
Output To Log File
```

Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the PP.

```
Parameter Name = Value
```

Name	string
Value	real

Parametric Search Tolerance

Set a parametric tolerance used to allow for points to be interpolated that lie just outside of an element.

```
Parametric Search Tolerance Tol
```

Tol	real

Pc Polynomial Degree

Specifies the global degree of the PC expansion.

```
Pc Polynomial Degree = poly_degree
```

poly_degree	integer

Polynomial Degree

Specify the polynomial degree to use for the recovery.

```
Polynomial Degree Degree
```

Degree	integer

Print Values Of

Prints the values of a field to the screen.

```
Print Values Of Option Field Field_name
```

Field_name	string

Process Initial Condition

This causes the first time step to be at time 0 when importing a solution... allowing post-processing and transfers of the initial condition.

It should probably only be used when running Encore as a standalone application, as it might throw other applications off.

```
Process Initial Condition
```

Random Field Basis Name

Specifies the name of the vector-valued field containing the random field basis vectors.

```
Random Field Basis Name = Rf_basis_name
```

Rf_basis_name	string

Random Field Basis Name

Specifies the name of the vector-valued field containing the random field basis vectors.

```
Random Field Basis Name = Rf_basis_name
```

Rf_basis_name	string

Random Field Name

Specifies the name of the field containing the random field realization.

```
Random Field Name = Rf_name
```

Rf_name	string

Random Field Name

Specifies the name of the field containing the random field realization.

```
Random Field Name = Rf_name
```

Rf_name	string

Random Variable Values

Specifies the vector of standard normal random variable values. These are used to evaluate the coefficient functions of the random field basis.

```
Random Variable Values = values…
```

values	real…

Random Variable Values

Specifies the vector of standard normal random variable values. These are used to evaluate the coefficient functions of the random field basis.

```
Random Variable Values = values…
```

values	real…

Ratio Is

Specify the Postprocessors to use in the ratio.

```
Ratio Is Postprocessor1 / Postprocessor2
```

Postprocessor1	string
Postprocessor2	string

Recover

Recovers the function in a specific way. If this line command is not present, the default will be to recover the Value.

```
Recover EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Rhs Postprocessor

Specify the name of the rhs postprocessor. Must be derived from SensitivePostprocessor.

```
Rhs Postprocessor Rhs_pp
```

Rhs_pp	string

Separator Character Is

This will cause the separator character specified to be used between output values of each postprocessor. The default is a comma.

```
Separator Character Is Exprs…
```

Exprs	(expression)…

Set Function Parameter

Sets the value of "param" on function "function_name" to the specified value.

```
Set Function Parameter Param_name On Function_name To Value
```

Param_name	string
Function_name	string
Value	real

Store Eigenvalues In

Specifies the name of the global variable in which to store the eigenvalues.

```
Store Eigenvalues In Eval_name
```

Eval_name	string

Store Eigenvectors In

Specifies the name of the field in which to store the eigenvectors.

```
Store Eigenvectors In Evec_name
```

Evec_name	string

Store In

Specify the name of the field in which you want to store the result.

```
Store In field_name
```

field_name	string

Store In

Specify the name of the field in which you want to store the result.

```
Store In field_name
```

field_name	string

Store In

Specify the name of the element field that will contain the mesh size values.

```
Store In Mesh_size_field
```

Mesh_size_field	string

Store In

Specify the name of the field in which you want to store the result.

```
Store In field_name
```

field_name	string

Store In

Specify a field in which to store the errors in the eigenfunctions.

```
Store In field_name
```

field_name	string

Store In

Specify the list of Fields in which to store the recovered values.

```
Store In recover_field_list…
```

recover_field_list	string…

Store In

Specify the name of the element field that will contain the mesh size values.

```
Store In MeshSizeField
```

MeshSizeField	string

String Parameter

Creates a parameter with the name and value specified. This parameter name can then be referenced by the PP.

```
String Parameter Name = Value
```

Name	string
Value	string

Subtract Function

Specifies the function to subtract from the other function for differences.

```
Subtract Function Function_name
```

Function_name	string

Surfaces

Specify all of the surfaces you wish to include in the postprocessor.

```
Surfaces Surface_list…
```

Surface_list	string…

Surfaces All

Specify to use all the surfaces in the postprocessor.

```
Surfaces All
```

Time

Evaluates the function at the given time. If not specified then the current simulation time is used instead. Note that this has no effect on field functions.

```
Time t
```

t	real

Track Point

Causes the actual place where the function gets evaluated to move with mesh motion.

```
Track Point
```

Upper Threshold

Restrict values counted to those less than or equal to this value.

```
Upper Threshold = x
```

x	real

Use Data Function

Specifies the data function to subtract from the use function.

```
Use Data Function Data_function_name
```

Data_function_name	string

Use Element Field

Specify the name of the element field that contains the error values.

```
Use Element Field Error_field
```

Error_field	string

Use Element Field

Specify the name of the integer element field that contains the marker values.

```
Use Element Field Marker_field
```

Marker_field	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specify a list of Functions.

```
Use Functions Function_list…
```

Function_list	string…

Use Internal Sierra Kl Data

Specifies to use Sierra internal fields and global variables rather than from input file.

```
Use Internal Sierra Kl Data
```

Use Internal Sierra Kl Data

Specifies to use Sierra internal fields and global variables rather than from input file.

```
Use Internal Sierra Kl Data
```

Use Postprocessor

Specifies the postprocessor to evaluate.

```
Use Postprocessor Pp_name
```

Pp_name	string

Use Weight Function

Specifies the weight function to use.

```
Use Weight Function Weight_Function
```

Weight_Function	string

Vector Parameter

Creates a parameter with the name and values specified. This parameter name can then be referenced by the PP.

```
Vector Parameter Name = Values…
```

Name	string
Values	real…

Verify Hanging Node Constraints

Verify hanging node constraints for nodal fields.

```
Verify Hanging Node Constraints
```

Volumes

Specify all of the volumes you wish to include in the postprocessor.

```
Volumes Volume_list…
```

Volume_list	string…

Volumes All

Specify to use all volumes in the postprocessor.

```
Volumes All
```

Write To File

Specifies the name of a file to write the results to.

```
Write To File Filename
```

Filename	string

Write To File

Specifies the name of a file to write the tabular results to.

```
Write To File Filename
```

Filename	string

Chapter 3

Transfers

In Sierra Mechanics, transfers are used to send data from one mesh to another. Transfers are typically used by application codes for code coupling, also known as operator splitting. For example, the thermal code Calore computes a temperature field at every node in the mesh. This temperature could determine a source term or boundary condition to the *Fuego* fluid flow application, or the *Adagio* solid mechanics application. Transfers are also often used in the context of code and solution verification. For verification, in order to assess the numerical error of a solution using a coarse grid, we would like to compare that solution with the solution on a fine grid.

Using Transfers

For transfers there is no requirement that the meshes be the same, or even that the meshes cover the same spatial domain. The Transfer is determined by

- the type of the data transfer: Copy or Interpolate,
- the sending and receiving meshes, or Regions,
- the mesh objects of the sending/receiving data, e.g. nodes or elements,
- the data fields to transfer.

There are also a number of other optional commands that can be used. For purposes of transfers, a Region is a mesh and the data fields defined on it.

Specifying a `Copy Transfer` should only be done when the sending and receiving meshes have exactly the same nodes and elements. In all other cases, users should specify `Interpolate` as the type of data transfer. This results in some kind of local interpolation to be used in order to calculate the data fields on the receiving mesh. For example, when transferring a nodal field to a nodal field, the finite element basis functions are used to interpolate the nodal values in the sending mesh to the locations of the nodal coordinates in the receiving mesh.

The sending data can be located at nodes, elements, or constraints; the receiving data can be located at nodes, elements, constraints, or integration points. Transfer to integration points is desirable for integrating a function on the receiving mesh that is defined using nodal fields and interpolation on the sending mesh. In this case, there is no error in evaluating the function on a different mesh, only through the numerical quadrature.

Transfers can also be performed between moving meshes. In this case, the deforming coordinate fields for each mesh must be specified using the `Search Coordinate Field` line command.

We now give some sample input syntax to illustrate some of the transfer capabilities that can be performed using Encore. We will assume throughout that there are two Regions that we denote by *`send_region`* and *`receive_region`*.

In order to use transfers, two entries must be included in the input file. The first entry is a `Begin/End Transfer` definition block, which is placed in the `Encore Procedure` block. The complete listing of all commands available in a `Transfer` block is included in the next section. The second entry is a line command to execute a `Transfer` in a `Solution Control` block (see also the chapter on `Solution Control`).

Suppose that we wish to interpolate a nodal field `X` defined in `send_region` into a nodal field `Y` defined in `receive_region`. For simplicity, assume that the problem is steady state, i.e., there is only one time step. The minimal syntax for the `Transfer` block is

```
Begin Transfer my_transfer
  Interpolate Volume Nodes From send_region To receive_region
  Send Field X State New To Y State New
End
```

The *`State`* of the sending and receiving fields is a result of internal storage of fields in Sierra Mechanics. Setting the state to *`New`* usually works correctly. Occasionally *`New`* will fail, however, if a field does not support multiple states. In this case, the state should be set to *`None`*.

Since Encore does not define any fields by default besides the coordinates field, both the sending and receiving fields, here *`X`* and *`Y`* need to be defined in their respective regions.

When comparing solutions on different meshes, the solution fields are typically read in from previously stored output files in the *ExodusII* format. In this case, we would add a line command inside the Region block of *`send_region`* such as

```
Import Field X_name As Nodal Field X
```

This loads a field name *`X_name`* from a previously written output file into a Sierra field named *`X`*. In the Region block of the *`receive_region`*, we would need to allocate a field to contain the transferred data *`Y`*. To do this, we would add a line command to *`receive_region`* such as

```
Create Nodal Field Y Of Type VECTOR_3D
```

The *`Type`* can be a scalar *`REAL`* or vector *`VECTOR_2D`*, etc., and it must match the type of the sending field.

Finally, in order to execute the transfer, include the *`Transfer`* line command into the *`Solution Control`* block. For our steady state example, this would be done like this

```
Begin Solution Control Description
  Use System main
  Begin System main
    Begin Transient encore_trans
```

```
      Advance send_region
      Transfer my_transfer
      Advance receive_region
    End
    Simulation Start Time = 0
    Simulation Termination Time = 1
    Simulation Max Global Iterations = 1
  End
End
```

Even though we use the *Transient* block, Encore will only take one time step, making the problem effectively steady state. Solution Control will execute the line commands in the order that they appear in the Transient block. The Advance command will execute whatever commands have been placed in the Region block. For our example, the first Advance command will cause the field X in send_region to be imported from the output database field X_name. Then the Transfer command will cause the field X to be transferred into the field Y in receive_region. Whereas the second Advance command will execute any commands in receive_region. These could include comparison of the transferred field Y with another field on receive_region, or the use of Y in a postprocessor that is computed on receive_region.

If we wished to interpolate an element field instead, in the code above we would replace Interpolate Volume Nodes with Interpolate Volume Elements. In the Elements transfer, the interpolation is based on a smoothing, or averaging, of the element centroid field values in the sending mesh. The values of elements in a patch around each node are fitted to a polynomial, which is evaluated at a centroid of an element in the receiving mesh. This feature is currently limited to meshes composed solely of hexahedral elements with eight nodes.

Transfers can also be used in Encore for true transient, or time-dependent, problems. You may wish to only use the set of time steps of only one of the input Regions involved in the transfer. In this case, use the line command Disable Compute Timestep in the Region you wish to ignore. You may also wish to delimit the output times, or compute on a subset of one of the input Region's timesteps. For an example of computing a transfer on a subset of the time interval, see Example 8 in the Appendix. Another useful command when performing transfers is the Process Initial Condition, which, if placed in the Region will include the initial state as one of the time steps in the transfer. Normally, by default, the initial state is ignored.

3.1 Transfer

transfer region/mesh information. the mechanics/variables information will get sorted out by the calling 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
   Experimental Option To Use Sending Mesh In Place
   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
```

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

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.

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

All Fields

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

```
All Fields
```

Copy

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.

```
Copy Option1 Option2 From From_region_name To To_region_name
```

From_region_name	string
To_region_name	string

Distance Function Is Closest Receive Node To Send Centroid

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.

```
Distance Function Is Closest Receive Node To Send Centroid
```

Exclude Ghosted

exclude ghosted nodes from a copy transfer

```
Exclude Ghosted
```

Experimental Option To Use Sending Mesh In Place

For testing purposes only. Normally the sending primary region's objects are copied to a secondary region. This can require a large amount of memory. If specified this line command will attempt to use the sending region objects without copying them. This does not always get the exact same answer as the standard method and I'm not sure why. It could be that just processing the mesh objects in a slightly different order causes slight differences or it could be that there is still a bug with this option.

```
Experimental Option To Use Sending Mesh In Place
```

From

Allows the send/receive mesh objects to be different.

```
From Option1 To Option2
```

Gauss Point Integration Order

Integration order to use when transferring to Gauss points.

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

Order	integer

Interpolate

Interpolate 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. There are many line commands that can be used to modify the behavior of the interpolation transfer but the basic algorithm is simple. 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 neighborhood of elements is found a bi-linear least squares fit is used to define the element field at the receiving point.

```
Interpolate Option1 Option2 From From_region_name To To_region_name
```

From_region_name	string
To_region_name	string

Interpolation Function

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.

```
Interpolation Function User_Subroutine
```

User_Subroutine	string

Nodes Outside Region

This line command defines what to do when a receiving point is outside the scope of the sending mesh. (1) 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. (2) To extrapolate 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. (3) Truncate is an option in which the receiving coordinate is projected back to the surface of the sending mesh to determine a value. This ensures that the receiving value is out outside of the field values in the sending mesh. (4) Projecte is an option similar to Trumcate 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.

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

Search Coordinate Field

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

```
Search Coordinate Field Source_field_name State Option1 To Destination_field_name State
   Option2
```

Source_field_name	string
Destination_field_nam	string

Search Geometric Tolerance

```
Search Geometric Tolerance {=|are|is} Geometric_tolerance
```

Geometric_tolerance	real

Search Surface Gap Tolerance

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 there will be a message printed about an "expensive search for extrapolation" that is done for these points. This 'expensive search" can be very expensive 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 is far too large for the rest of the mesh. Setting "or less' lets the search reduce the tolerance in areas of the mesh resulting in a faster search.

```
Search Surface Gap Tolerance {=|are|is} Surface_gap_tolerance [ Or Less ]
```

Surface_gap_tolerance	real

Search Type

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

Select One Receiver For Each Send Object

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.

```
Select One Receiver For Each Send Object
```

Select One Unique Receiver For Each Send Object

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.

```
Select One Unique Receiver For Each Send Object
```

Send

Use predefine transfer semantics provided by the specified name.

```
Send Predefined-transfer Fields
```

Predefined-transfer	{}

Send Block

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

```
Send Block From_blocks… To To_blocks…
```

From_blocks	string…
To_blocks	string…

Send Field

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.

```
Send Field Source_field_name State Option1 To Destination_field_name State Option2 [ Lower
   Bound Lower_bound Upper Bound Upper_bound ]
```

Source_field_name	string
Destination_field_nam	string

Notes on subscripting:

1. Does not work for COPY transfers, only INTERPOLATION type transfers.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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 4

Indicator Commands

4.1 Time Integral Indicator

This indicator takes the element field computed from another indicator and integrates the elements values over time.

```
Begin Time Integral Indicator Name
   Error Values Are Signed
   Store In Field_name
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Error Values Are Signed

Error values can be positive or negative.

```
Error Values Are Signed
```

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

4.2 Function Indicator

Contains the commands needed to specify an indicator that evaluates a given function on the element and pushes that into an element field.

```
Begin Function Indicator Name
   Store In Field_name
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

4.3 Adjoint Indicator

todo

```
Begin Adjoint Indicator Name
   Store In Field_name
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

4.4 Sierra Realize Random Field Error Indicator

todo

```
Begin Sierra Realize Random Field Error Indicator Name
   Random Variable Values = values…
   Store In Field_name
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Random Variable Values

Specifies the vector of standard normal random variable values. These are used to evaluate the coefficient functions of the random field basis.

```
Random Variable Values = values…
```

values	real…

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

4.5 Jump Indicator

Contains the commands needed to specify an indicator that computes the jumps in the normal component of the gradient of a function across element interfaces.

```
Begin Jump Indicator Name
   Calculate Jump In EvalFunctionTypes
   Store In Field_name
   Use Adjoint Function Adjoint_function_list…
   Use Adjoint Functions Adjoint_function_list…
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Calculate Jump In

Specifies the function method that will be called in the evaluation of the jump.

```
Calculate Jump In EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Adjoint Function

Specifies the list of adjoint functions.

```
Use Adjoint Function Adjoint_function_list…
```

Adjoint_function_list	string…

Use Adjoint Functions

Specifies the list of adjoint functions.

```
Use Adjoint Functions Adjoint_function_list…
```

Adjoint_function_list	string…

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

4.6 Recovery Indicator

Contains the commands needed to specify an indicator that recovers the given function then stores the difference between the recovery and the function on each element.

```
Begin Recovery Indicator Name
   Polynomial Degree Degree
   Recover EvalFunctionTypes
   Store In Field_name
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Polynomial Degree

Specify the polynomial degree to use for the recovery.

```
Polynomial Degree Degree
```

Degree	integer

Recover

Recovers the function in a specific way. If this line command is not present, the default will be to recover the Value.

```
Recover EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

4.7 Meta Indicator

Contains the commands needed to specify an indicator that derives its values through summing up the output of multiple indicators.

```
Begin Meta Indicator Name
   Store In Field_name
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
   Use Indicators Indicators…
End
```

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

Use Indicators

Specifies the indicators whose values will be summed.

```
Use Indicators Indicators…
```

Indicators	string…

4.8 Curvature Indicator

Contains the commands needed to specify an indicator that indicates where surfaces have high curvature.

```
Begin Curvature Indicator Name
   Averaging Is least squares|simple average
   Store In Field_name
   Surfaces Surface_list…
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Averaging Is

Specifies what kind of averaging to use.

```
Averaging Is least squares|simple average
```

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Surfaces

Specify all of the surfaces you wish to include.

```
Surfaces Surface_list…
```

Surface_list	string…

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

4.9 Block Indicator

This indicator takes the element field computed from another indicator and sums the values over the element blocks.

```
Begin Block Indicator Name
   Store In Field_name
   Use Element Field Use_field_name
   Use Function Function_list
   Use Functions Function_list…
End
```

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

Averaging Is

Specifies what kind of averaging to use.

```
Averaging Is least squares|simple average
```

Calculate Jump In

Specifies the function method that will be called in the evaluation of the jump.

```
Calculate Jump In EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Polynomial Degree

Specify the polynomial degree to use for the recovery.

```
Polynomial Degree Degree
```

Degree	integer

Random Variable Values

Specifies the vector of standard normal random variable values. These are used to evaluate the coefficient functions of the random field basis.

```
Random Variable Values = values…
```

values	real…

Recover

Recovers the function in a specific way. If this line command is not present, the default will be to recover the Value.

```
Recover EvalFunctionTypes
```

EvalFunctionTypes	{dot | flux | gradient | stress | track | value}

Store In

Specifies the element field name to store the value in.

```
Store In Field_name
```

Field_name	string

Surfaces

Specify all of the surfaces you wish to include.

```
Surfaces Surface_list…
```

Surface_list	string…

Use Adjoint Function

Specifies the list of adjoint functions.

```
Use Adjoint Function Adjoint_function_list…
```

Adjoint_function_list	string…

Use Adjoint Functions

Specifies the list of adjoint functions.

```
Use Adjoint Functions Adjoint_function_list…
```

Adjoint_function_list	string…

Use Element Field

Specify an input element indicator field.

```
Use Element Field Use_field_name
```

Use_field_name	string

Use Function

Specifies the function to evaluate.

```
Use Function Function_list
```

Function_list	string

Use Functions

Specifies the function to evaluate.

```
Use Functions Function_list…
```

Function_list	string…

Use Indicators

Specifies the indicators whose values will be summed.

```
Use Indicators Indicators…
```

Indicators	string…

Chapter 5

Marker Commands

5.1 Statistical Marker

Contains the commands needed to specify a marker that marks based on a number of standard deviations away from the mean.

```
Begin Statistical Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Error Values Are Signed
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Number Of Standard Deviations To Coarsen Below Value
   Number Of Standard Deviations To Refine Above Value
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Error Values Are Signed

Error values can be positive or negative.

```
Error Values Are Signed
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Number Of Standard Deviations To Coarsen Below

This equates to refining all elements whose use field value is below the mean()-(value * standard_deviation).

Remember from statistics class that (from Wikipedia): "About 68

But since we are only taking the elements that lie to the right of the standard deviations plus the mean that means that if the error distribution is normal (Gaussian) and the value to this line command is 0 you will be coarsening 50

```
Number Of Standard Deviations To Coarsen Below Value
```

Value	real

Number Of Standard Deviations To Refine Above

This equates to refining all elements whose use field value is above the mean()+(value * standard_deviation).

Remember from statistics class that (from Wikipedia): "About 68

But since we are only taking the elements that lie to the right of the standard deviations plus the mean that means that if the error distribution is normal (Gaussian) and the value to this line command is 0 you will be refining 50

```
Number Of Standard Deviations To Refine Above Value
```

Value	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.2 Reentrant Corner Marker

Contains the commands needed to specify a marker that marks all reentrant corners.

```
Begin Reentrant Corner Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.3 Above Below Marker

Contains the commands needed to specify a marker that marks elements whose "Use Field" is above "Refine Above" for refinement and whose value is below "Coarsen Below" for coarsening.

```
Begin Above Below Marker Name
   Coarsen Below Value
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Refine Above Value
   Store In field_name
   Use Option Field Field_name
End
```

Coarsen Below

Specifies the threshold value for coarsening.

```
Coarsen Below Value
```

Value	real

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Refine Above

Specifies the threshold value for refinement.

```
Refine Above Value
```

Value	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.4 Meta Marker

Contains the commands needed to specify a marker that marks based on the output of other markers.

```
Begin Meta Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Store In field_name
   Use Option Field Field_name
   Use Markers Marker_list…
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

Use Markers

A list of markers to combine.

```
Use Markers Marker_list…
```

Marker_list	string…

5.5 Exposed Boundary Marker

Contains the commands needed to specify a marker that marks all exposed boundary elements for refinement.

```
Begin Exposed Boundary Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.6 Interval Marker

Contains the commands needed to specify a marker that marks elements whose "Use Field" is within specified limits.

```
Begin Interval Marker Name
   Coarsen Between Lower Upper
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Refine Between Lower Upper
   Store In field_name
   Use Option Field Field_name
End
```

Coarsen Between

Specifies the threshold values for coarsening.

```
Coarsen Between Lower Upper
```

Lower	real
Upper	real

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Refine Between

Specifies the threshold values for refinement.

```
Refine Between Lower Upper
```

Lower	real
Upper	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.7 Nodeset Marker

Contains the commands needed to specify a marker that marks elements with atleast one side in the sideset.

```
Begin Nodeset Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Nodesets Meshpart_list…
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Nodesets

Specify all of the nodesets you wish to mark.

```
Nodesets Meshpart_list…
```

Meshpart_list	string…

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.8 Percent Of Total Error Marker

Contains the commands needed to specify a marker that marks a percentage of the sum of the use field values.

```
Begin Percent Of Total Error Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Percent Of Total Error To Coarsen Value
   Percent Of Total Error To Refine Value
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Percent Of Total Error To Coarsen

Specifies the percent of the sum of the use field values to coarsen.

```
Percent Of Total Error To Coarsen Value
```

Value	real

Percent Of Total Error To Refine

Specifies the percent of the sum of the use field values to refine.

```
Percent Of Total Error To Refine Value
```

Value	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.9 Surface Marker

Contains the commands needed to specify a marker that marks elements with atleast one side in the sideset.

```
Begin Surface Marker Name
   Cutoff Sensitivity Value
```

```
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Store In field_name
   Surfaces Meshpart_list…
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Surfaces

Specify all of the surfaces you wish to mark.

```
Surfaces Meshpart_list…
```

Meshpart_list	string…

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.10 Volume Marker

Contains the commands needed to specify a marker that marks specified volumes.

```
Begin Volume Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Store In field_name
   Use Option Field Field_name
   Volumes Volume_list…
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

Volumes

Specify all of the volumes you wish to mark.

```
Volumes Volume_list…
```

Volume_list	string…

5.11 Percent Of Elements Marker

Contains the commands needed to specify a marker that marks a percentage of elements for refinement and coarsening.

```
Begin Percent Of Elements Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Percent Of Elements To Coarsen Value
   Percent Of Elements To Refine Value
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Percent Of Elements To Coarsen

Specifies the percent of elements to mark for coarsening.

```
Percent Of Elements To Coarsen Value
```

Value	real

Percent Of Elements To Refine

Specifies the percent of elements to mark for refinement.

```
Percent Of Elements To Refine Value
```

Value	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.12 Block Boundary Marker

Contains the commands needed to specify a marker that marks all block boundary elements for refinement.

```
Begin Block Boundary Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.13 Bounding Box Marker

Contains the commands needed to mark elements with centroids that intersect a bounding box.

```
Begin Bounding Box Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Refine Within Box_min1[ Box_min2[ Box_min3]] And Box_max1[ Box_max2[ Box_max3]]
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Refine Within

Mark elements for refinement that have a centroid that intersects a box defined by min and max coordinates. The box is aligned with the Cartesion x,y,z axes. The min point must have x,y,z coordinates that are less than the max point x,y,z coordinates.

```
Refine Within Box_min1[ Box_min2[ Box_min3]] And Box_max1[ Box_max2[ Box_max3]]
```

Box_min	real_1[real_2[real_3]]
Box_max	real_1[real_2[real_3]]

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.14 Percent Of Max Marker

Contains the commands needed to specify a marker that marks elements whose "Use Field" is above "Refine Above" for refinement and whose value is below "Coarsen Below" for coarsening.

```
Begin Percent Of Max Marker Name
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Percent Of Max To Coarsen Below Value
   Percent Of Max To Refine Above Value
   Store In field_name
   Use Option Field Field_name
End
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Percent Of Max To Coarsen Below

Specifies the threshold value for coarsening.

```
Percent Of Max To Coarsen Below Value
```

Value	real

Percent Of Max To Refine Above

Specifies the threshold value for refinement.

```
Percent Of Max To Refine Above Value
```

Value	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

5.15 Uniform Marker

Contains the commands needed to specify a marker that marks all elements for refinement.

```
Begin Uniform Marker Name
   Coarsen
   Cutoff Sensitivity Value
   Disable Coarsening
   Maximum Number Of Elements Num
   Maximum Refinement Level Level
   Minimum Element Size Size
   Store In field_name
   Use Option Field Field_name
End
```

Coarsen

Mark all elements for coarsening (instead of the default of refining).

```
Coarsen
```

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

Coarsen

Mark all elements for coarsening (instead of the default of refining).

```
Coarsen
```

Coarsen Below

Specifies the threshold value for coarsening.

```
Coarsen Below Value
```

Value	real

Coarsen Between

Specifies the threshold values for coarsening.

```
Coarsen Between Lower Upper
```

Lower	real
Upper	real

Cutoff Sensitivity

Specifies the cutoff sensitivity used in markers.

```
Cutoff Sensitivity Value
```

Value	real

Disable Coarsening

Do not allow any coarsening. This is useful when running multiple Markers.

```
Disable Coarsening
```

Error Values Are Signed

Error values can be positive or negative.

```
Error Values Are Signed
```

Maximum Number Of Elements

Specifies the maximum number of elements used for refinement marking.

```
Maximum Number Of Elements Num
```

Num	integer

Maximum Refinement Level

Specifies the maximum refinement level used for refinement marking.

```
Maximum Refinement Level Level
```

Level	integer

Minimum Element Size

Specifies the minimum element size used for refinement marking.

```
Minimum Element Size Size
```

Size	real

Nodesets

Specify all of the nodesets you wish to mark.

```
Nodesets Meshpart_list…
```

Meshpart_list	string…

Number Of Standard Deviations To Coarsen Below

This equates to refining all elements whose use field value is below the mean()-(value * standard_deviation).

Remember from statistics class that (from Wikipedia): "About 68

But since we are only taking the elements that lie to the right of the standard deviations plus the mean that means that if the error distribution is normal (Gaussian) and the value to this line command is 0 you will be coarsening 50

```
Number Of Standard Deviations To Coarsen Below Value
```

Value	real

Number Of Standard Deviations To Refine Above

This equates to refining all elements whose use field value is above the mean()+(value * standard_deviation).

Remember from statistics class that (from Wikipedia): "About 68

But since we are only taking the elements that lie to the right of the standard deviations plus the mean that means that if the error distribution is normal (Gaussian) and the value to this line command is 0 you will be refining 50

```
Number Of Standard Deviations To Refine Above Value
```

Value	real

Percent Of Elements To Coarsen

Specifies the percent of elements to mark for coarsening.

```
Percent Of Elements To Coarsen Value
```

Value	real

Percent Of Elements To Refine

Specifies the percent of elements to mark for refinement.

```
Percent Of Elements To Refine Value
```

Value	real

Percent Of Max To Coarsen Below

Specifies the threshold value for coarsening.

```
Percent Of Max To Coarsen Below Value
```

Value	real

Percent Of Max To Refine Above

Specifies the threshold value for refinement.

```
Percent Of Max To Refine Above Value
```

Value	real

Percent Of Total Error To Coarsen

Specifies the percent of the sum of the use field values to coarsen.

```
Percent Of Total Error To Coarsen Value
```

Value	real

Percent Of Total Error To Refine

Specifies the percent of the sum of the use field values to refine.

```
Percent Of Total Error To Refine Value
```

Value	real

Refine Above

Specifies the threshold value for refinement.

```
Refine Above Value
```

Value	real

Refine Between

Specifies the threshold values for refinement.

```
Refine Between Lower Upper
```

Lower	real
Upper	real

Refine Within

Mark elements for refinement that have a centroid that intersects a box defined by min and max coordinates. The box is aligned with the Cartesion x,y,z axes. The min point must have x,y,z coordinates that are less than the max point x,y,z coordinates.

```
Refine Within Box_min1[ Box_min2[ Box_min3]] And Box_max1[ Box_max2[ Box_max3]]
```

Box_min	real_1[real_2[real_3]]
Box_max	real_1[real_2[real_3]]

Store In

Specifies the element field name to store the value in.

```
Store In field_name
```

field_name	string

Surfaces

Specify all of the surfaces you wish to mark.

```
Surfaces Meshpart_list…
```

Meshpart_list	string…

Use

Specifies the field to use for the values.

```
Use Option Field Field_name
```

Field_name	string

Use Markers

A list of markers to combine.

```
Use Markers Marker_list…
```

Marker_list	string…

Volumes

Specify all of the volumes you wish to mark.

```
Volumes Volume_list…
```

Volume_list	string…

Chapter 6

Domain and Procedure Level Commands

6.1 Sierra

Delimits a domain to contain the procedures.

```
Begin Sierra JobIdentifier
   Load User Plugin File File Name [ Using Function Function ]
   Restart {=|are|is} Option
   Restart Time {=|are|is} Time
   Serialized Io Group Size {=|are|is} Group-size
   Test Error Messages To File File_name And Die On First types…
   Title
   User Subroutine File {=|are|is} File_name
   Log Log-control-name Every Interval
   Print Timer Information Every Procedure-step-interval Steps Checkpointed
   Begin Above Below Marker Name
   End
   Begin Adjoint Indicator Name
   End
   Begin Adjoint Test Postprocessor Postprocessor_name
   End
   Begin Average Value Postprocessor Postprocessor_name
   End
   Begin Block Boundary Marker Name
   End
   Begin Block Indicator Name
   End
   Begin Bounding Box Marker Name
   End
   Begin Curvature Indicator Name
   End
   Begin Difference Function Function_name
   End
```

```
Begin Difference Postprocessor Postprocessor_name
End
Begin Dakota
End
Begin Encore Procedure ProcedureName
End
Begin Equivalent Cure Postprocessor Postprocessor_name
End
Begin Evaluate Function Postprocessor Postprocessor_name
End
Begin Exposed Boundary Marker Name
End
Begin Fad User Function Function_name
End
Begin Field Function Function_name
End
Begin Finite Element Model Label
End
Begin Fortran User Function Function_name
End
Begin Function Indicator Name
End
Begin Global Function Parameters Gufp
End
Begin Integral Least Squares Difference Postprocessor Postprocessor_name
End
Begin Integrate Function Postprocessor Postprocessor_name
End
Begin Interval Marker Name
End
Begin Jump Indicator Name
End
Begin Least Squares Difference Postprocessor Postprocessor_name
End
Begin Mesh Size From Error Postprocessor_name
End
Begin Mesh Size From Marker Postprocessor_name
End
Begin Meta Indicator Name
End
Begin Meta Marker Name
End
Begin Min Max Postprocessor Postprocessor_name
End
Begin Nodeset Marker Name
End
Begin Norm Postprocessor Postprocessor_name
End
Begin Output Scheduler Label
End
Begin Percent Of Elements Marker Name
End
Begin Percent Of Max Marker Name
End
Begin Percent Of Total Error Marker Name
```

```
End
Begin Percent Threshold Postprocessor Postprocessor_name
End
Begin Postprocessor Output Control Name
End
Begin Product Function Function_name
End
Begin Quadratic Difference Postprocessor Postprocessor_name
End
Begin Ratio Postprocessor Postprocessor_name
End
Begin Realize Pc Random Field Name
End
Begin Realize Random Field Name
End
Begin Recover Function Postprocessor Postprocessor_name
End
Begin Recovery Indicator Name
End
Begin Reentrant Corner Marker Name
End
Begin Sierra Kl Solver Name
End
Begin Sierra Kl Solver Error Indicator Name
End
Begin Sierra Realize Random Field Error Indicator Name
End
Begin Statistical Marker Name
End
Begin String Function Function_name
End
Begin Summation Postprocessor Postprocessor_name
End
Begin Surface Marker Name
End
Begin Surface Normal Postprocessor Postprocessor_name
End
Begin Tabular Function Output Postprocessor Postprocessor_name
End
Begin Time Integral Indicator Name
End
Begin Title Title
End
Begin Uniform Marker Name
End
Begin Unit Normal Function Function_name
End
Begin User Function Function_name
End
Begin User Postprocessor Function_name
End
Begin Vector Function Function_name
End
Begin Volume Marker Name
End
```

```
End
```

The "Begin Sierra jobid" and "End Sierra jobid" block contains the input commands for the analysis run. The jobid is an arbitray string identifying the analysis

Load User Plugin File

This line command names the source file and registration function for C++ user plugins, subroutines and functions.

```
Load User Plugin File File Name [ Using Function Function ]
```

File Name	string

Restart

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.

```
Restart {=|are|is} Option
```

Restart Time

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.

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

Time	real

Serialized Io Group Size

Specifies the number of processors which can concurrently perform I/O. Specifying zero disables serialization.

```
Serialized Io Group Size {=|are|is} Group-size
```

Group-size	integer

Test Error Messages To File

Write a error messageS to the specified file and then die

```
Test Error Messages To File File_name And Die On First types…
```

File_name	string

Title

User-defined title for identifying the analysis. The title continues to the end of the line (including continuation lines)

```
Title
```

User Subroutine File

This line command is really only for the script that runs calore/fuego/premo... It needs to know where to find the *.F file that contains all the user subroutines that are referenced in the input file. Although a C++ handler is provided, the apps do not do anything with this command, only the script that runs the app. Note that the scope of this command is domain!

```
User Subroutine File {=|are|is} File_name
```

File_name	string

Log

Sets the maximum number of warnings before the execution is terminated.

```
Log Log-control-name Every Interval
```

Log-control-name	string
Interval	integer

Print Timer Information Every

Specifies the procedure step count interval to print timer information

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

Procedure-step-interval	integer
Checkpointed	{accumulated|checkpointed}

6.2 Encore Procedure

The Encore procedure is the second highest level block in the input file description hierarchy. It is required and contains the description blocks for the calore region and time control

```
Begin Encore Procedure ProcedureName
   Begin Encore Region Region_name
   End
   Begin Solution Control Description Name
   End
   Begin Transfer Transfer_name
   End
End
```

This block command is used to contain Encore commands that are associated with a solution procedure defined for a set of Encore regions. The name of the procedure is specified by the user. Note that the Encore Procedure command block must be present in the input file and must contain at least one ENCORE REGION command block. There are currently no line commands available for the PROCEDURE scope.

Chapter 7

Region Level Commands

7.1 Encore Region

The Encore Region

```
Begin Encore Region Region_name
   Advance To Final Time
   Compute Difference NormTypes Of f1 f2 [ Store In field_name ]
   Compute Norm NormTypes Of Function_name [ Store In field_name ]
   Constant Timestep Is dt
   Create Edge Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
   Create Element Field Field_name Of Type Option And Dimension Dimension [ On Part_list…
      ]
   Create Face Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
   Create Nodal Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
   Create Quadrature Field Field_name Of Type Option And Order Intg_order
   Current Coordinates Are Field_name
   Disable Compute Timestep
   Displace Coordinates Using Displacements_field_name Into Displaced_field_name
   Enable Rebalance [ Using Zoltan Parameters Name ]
   Evaluate Flux Of Function_name At x [ y z t ]
   Evaluate Function Function_name At x [ y z t ]
   Evaluate Gradient Of Function_name At x [ y z t ]
   Evaluate Postprocessor Postprocessor_name
   Evaluate Time Derivative Of Function_name At x [ y z t ]
   Execute Postprocessor Group Postprocessor_name
   Import Field Mesh_field_name As Option1 Field Field_name [ Of Type Option2 ]
   Indicatemarkadapt Using Indicator Marker
   Initial Markadapt Using Marker For Num Iterations
   Interpolate Function Option1 Of Function_name Into Option2 Field Field_name [ On Op-
      tion3 Part_list… ]
   Markadapt Using Marker
   Output Indicator Global Value Of Indicator_name
   Output Number Of Option
   Output Postprocessor History Of Pp_name
```

```
   Output Preferred Integration Order Of Function_name
   Print Values Of Option Field Field_name
   Process Initial Condition
   Set Function Parameter Param_name On Function_name To Value
   Use Finite Element Model ModelName [ Model Coordinates Are Nodal_variable_name ]
   Verify Hanging Node Constraints
   Begin Geometry GeometryName
   End
   Begin Heartbeat Label
   End
   Begin History Output Label
   End
   Begin Postprocessor Group Group_name
   End
   Begin Restart Data Label
   End
   Begin Results Output Label
   End
End
```

This block command is used to contain Encore region commands.

Advance To Final Time

Causes Encore to take a single time step ending in the last time stored in the input mesh file. If no time steps are present, does nothing.

```
Advance To Final Time
```

Compute Difference

Computes the error between the two functions using the specified norm.
In the case of relative norms it does norm(f1-f2) / norm(f1) .

```
Compute Difference NormTypes Of f1 f2 [ Store In field_name ]
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}
f1	string
f2	string

Compute Norm

Computes the norm of the function using the specified norm.

```
Compute Norm NormTypes Of Function_name [ Store In field_name ]
```

NormTypes	{h1|h1 restriction|l1|l2|linfinity|nodal l1|nodal l2|nodal linfinity|relative h1|relative h1 restriction|relative l1|relative l2|relative linfinity|relative nodal l1|relative nodal l2|relative nodal linfinity}
Function_name	string

Constant Timestep

Sets a constant time step for the Code region. This value is ignored if the time step is chosen from the input mesh.

```
Constant Timestep Is dt
```

dt	real

Create Edge Field

Creates an edge Field named field_name on the region.

```
Create Edge Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Element Field

Creates an Element Field name field_name on the region.

```
Create Element Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Face Field

Creates a face Field named field_name on the region.

```
Create Face Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Nodal Field

Creates a Nodal Field name field_name on the region.

```
Create Nodal Field Field_name Of Type Option And Dimension Dimension [ On Part_list… ]
```

Field_name	string
Dimension	integer

Create Quadrature Field

Creates an Element Field named field_name sized correctly to receive a quadrature transfer of the order specified.

```
Create Quadrature Field Field_name Of Type Option And Order Intg_order
```

Field_name	string
Intg_order	integer

Current Coordinates Are

This will cause an Encore region to use this field as it's "current coordinates" when doing things like integrating and such.

```
Current Coordinates Are Field_name
```

Field_name	string

Disable Compute Timestep

Disables the regions participation in timestep computation.

```
Disable Compute Timestep
```

Displace Coordinates Using

Causes Encore to use displaced_field_name as the current_coordinates... which is calculated by model_coordinates + displacements_field_name

```
Displace Coordinates Using Displacements_field_name Into Displaced_field_name
```

Displacements_field_n	string
Displaced_field_name	string

Enable Rebalance

Enable dynamic load balancing (rebalance). Optional command defines the name of the Zoltan parameters block used in rebalancing. Default value is "DEFAULT."

```
Enable Rebalance [ Using Zoltan Parameters Name ]
```

Evaluate Flux Of

Evaluates the gradient of the function at the given point.

```
Evaluate Flux Of Function_name At x [ y z t ]
```

Function_name	string
x	real

Evaluate Function

Evaluates the function at the given point.

```
Evaluate Function Function_name At x [ y z t ]
```

Function_name	string
x	real

Evaluate Gradient Of

Evaluates the gradient of the function at the given point.

```
Evaluate Gradient Of Function_name At x [ y z t ]
```

Function_name	string
x	real

Evaluate Postprocessor

Evaluates the specified processor at the current time.

```
Evaluate Postprocessor Postprocessor_name
```

Postprocessor_name	string

Evaluate Time Derivative Of

Evaluates the time derivative of the function at the given point.

```
Evaluate Time Derivative Of Function_name At x [ y z t ]
```

Function_name	string
x	real

Execute Postprocessor Group

Executes all of the postprocessors in the specified group.

```
Execute Postprocessor Group Postprocessor_name
```

Postprocessor_name	string

Import Field

Imports a field from the current mesh and copies it to a SIERRA field. The field must be nodal or element. The optional parameter is the time at which to read the field.

```
Import Field Mesh_field_name As Option1 Field Field_name [ Of Type Option2 ]
```

Mesh_field_name	string
Field_name	string

Indicatemarkadapt Using

Postprocessor that simulates the solution control IndicateMarkAdapt line command. When this Postprocessor executes the region will be immediately adapted once. To get more complicated execution (for instance more levels of refinement) you must use Solution Control.

```
Indicatemarkadapt Using Indicator Marker
```

Indicator	string
Marker	string

Initial Markadapt Using

Causes the marker listed to be used for num iterations and an adaptive step done on the Region before the first execution of the Region. This also precedes initialization of the Region, including setting initial conditions.

```
Initial Markadapt Using Marker For Num Iterations
```

Marker	string
Num	integer

Interpolate Function

Evaluates the specified function at the nodal positions and stores the result in field_name. Optionally, a list of surfaces or volumes can be listed that restrict the interpolation.

```
Interpolate Function Option1 Of Function_name Into Option2 Field Field_name [ On Option3
   Part_list… ]
```

Function_name	string
Field_name	string

Markadapt Using

Postprocessor that simulates the solution control MarkAdapt line command. When this Postprocessor executes the region will be immediately adapted once. To get more complicated execution (for instance more levels of refinement) you must use Solution Control.

```
Markadapt Using Marker
```

Marker	string

Output Indicator Global Value Of

Outputs the global value of an indicator. This is usually the summation of each element's error contribution.

```
Output Indicator Global Value Of Indicator_name
```

Indicator_name	string

Output Number Of

A postprocessor that just outputs the current number of active nodes or elements.

```
Output Number Of Option
```

Output Postprocessor History Of

Outputs the cumulative PP history values.

```
Output Postprocessor History Of Pp_name
```

Pp_name	string

Output Preferred Integration Order Of

Outputs the preferred integration order of the function.

```
Output Preferred Integration Order Of Function_name
```

Function_name	string

Print Values Of

Prints the values of a field to the screen.

```
Print Values Of Option Field Field_name
```

Field_name	string

Process Initial Condition

This causes the first time step to be at time 0 when importing a solution... allowing post-processing and transfers of the initial condition.

It should probably only be used when running Encore as a standalone application, as it might throw other applications off.

```
Process Initial Condition
```

Set Function Parameter

Sets the value of "param" on function "function_name" to the specified value.

```
Set Function Parameter Param_name On Function_name To Value
```

Param_name	string
Function_name	string
Value	real

Use Finite Element Model

Associates a predefined finite element model with this region.

```
Use Finite Element Model ModelName [ Model Coordinates Are Nodal_variable_name ]
```

ModelName	string

Verify Hanging Node Constraints

Verify hanging node constraints for nodal fields.

```
Verify Hanging Node Constraints
```

Constant Timestep

Sets a constant time step for the Code region. This value is ignored if the time step is chosen from the input mesh.

```
Constant Timestep Is dt
```

dt	real

Disable Compute Timestep

Disables the regions participation in timestep computation.

```
Disable Compute Timestep
```

Enable Rebalance

Enable dynamic load balancing (rebalance). Optional command defines the name of the Zoltan parameters block used in rebalancing. Default value is "DEFAULT."

```
Enable Rebalance [ Using Zoltan Parameters Name ]
```

Chapter 8

Solution Control Commands

8.1 Adaptivity

This block is used to wrap an adapt loop.

```
Begin Adaptivity 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 ]
   Mark Region_name… Using Marker_name… [ When When-expression ]
   Markadapt Region_name Using Marker [ When When-expression ]
   Transfer Name [ When When-expression ]
   Begin Sequential Name
   End
   Begin Transient Name
   End
End
```

Adapt

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

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

Region_name	string…
Field_name	string…

Advance

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

```
Advance Name… [ When When-expression ]
```

Name	string…

Compute Indicator On

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

```
Compute Indicator On Region_name… Using Indicator_name… [ When When-expression ]
```

Region_name	string…
Indicator_name	string…

Event

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.

```
Event Name… [ When When-expression ]
```

Name	string…

Execute Postprocessor Group

Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

```
Execute Postprocessor Group Group_name… On Region_name… [ When When-expression ]
```

Group_name	string…
Region_name	string…

Indicatemarkadapt

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

```
Indicatemarkadapt Region_name Using Indicator Marker [ When When-expression ]
```

Region_name	string
Indicator	string
Marker	string

Mark

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

```
Mark Region_name… Using Marker_name… [ When When-expression ]
```

Region_name	string…
Marker_name	string…

Markadapt

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

```
Markadapt Region_name Using Marker [ When When-expression ]
```

Region_name	string
Marker	string

Transfer

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.

```
Transfer Name [ When When-expression ]
```

Name	string

8.2 Solution Control Description

Contains the commands needed to execute an analysis using the Calagio procedure that utilizes Solver Control.

```
Begin Solution Control Description Name
   Use System Name
   Begin Initialize Name
   End
   Begin Parameters For
   End
   Begin System Name
   End
End
```

Use System

This set the name of which system to use.

```
Use System Name
```

Name	string

8.3 System

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.

```
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
```

Adapt

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

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

Region_name	string…
Field_name	string…

Compute Indicator On

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

```
Compute Indicator On Region_name… Using Indicator_name… [ When When-expression ]
```

Region_name	string…
Indicator_name	string…

Event

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.

```
Event Name… [ When When-expression ]
```

Name	string…

Execute Postprocessor Group

Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

```
Execute Postprocessor Group Group_name… On Region_name… [ When When-expression ]
```

Group_name	string…
Region_name	string…

Indicatemarkadapt

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

```
Indicatemarkadapt Region_name Using Indicator Marker [ When When-expression ]
```

Region_name	string
Indicator	string
Marker	string

Mark

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

```
Mark Region_name… Using Marker_name… [ When When-expression ]
```

Region_name	string…
Marker_name	string…

Markadapt

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

```
Markadapt Region_name Using Marker [ When When-expression ]
```

Region_name	string
Marker	string

Output

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

```
Output Name [ When When-expression ]
```

Name	string

Simulation Max Global Iterations

The Total number of Solves.

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

Number	integer

Simulation Start Time

Simulation starting time. (by default 0.0)

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

Number	real

Simulation Termination Time

The drop dead time.

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

Number	real

Transfer

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.

```
Transfer Name [ When When-expression ]
```

Name	string

Use Initialize

This set the name of which initialization to use.

```
Use Initialize Name
```

Name	string

8.4 Transient

This block is used to wrap a time loop.

```
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
```

Adapt

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

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

Region_name	string…
Field_name	string…

Advance

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

```
Advance Name… [ When When-expression ]
```

Name	string…

Compute Indicator On

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

```
Compute Indicator On Region_name… Using Indicator_name… [ When When-expression ]
```

Region_name	string…
Indicator_name	string…

Event

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.

```
Event Name… [ When When-expression ]
```

Name	string…

Execute Postprocessor Group

Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

```
Execute Postprocessor Group Group_name… On Region_name… [ When When-expression ]
```

Group_name	string…
Region_name	string…

Indicatemarkadapt

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

```
Indicatemarkadapt Region_name Using Indicator Marker [ When When-expression ]
```

Region_name	string
Indicator	string
Marker	string

Involve

Specifiy a physics participant to a coupled problem solved using matrix-free nonlinear.

```
Involve Name
```

Name	string

Mark

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

```
Mark Region_name… Using Marker_name… [ When When-expression ]
```

Region_name	string…
Marker_name	string…

Markadapt

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

```
Markadapt Region_name Using Marker [ When When-expression ]
```

Region_name	string
Marker	string

Output

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

```
Output Name [ When When-expression ]
```

Name	string

Transfer

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.

```
Transfer Name [ When When-expression ]
```

Name	string

8.5 Nonlinear

This block is used to wrap a nonlinear solve loop.

```
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
```

Adapt

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

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

Region_name	string…
Field_name	string…

Advance

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

```
Advance Name… [ When When-expression ]
```

Name	string…

Compute Indicator On

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

```
Compute Indicator On Region_name… Using Indicator_name… [ When When-expression ]
```

Region_name	string…
Indicator_name	string…

Event

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.

```
Event Name… [ When When-expression ]
```

Name	string…

Execute Postprocessor Group

Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

```
Execute Postprocessor Group Group_name… On Region_name… [ When When-expression ]
```

Group_name	string…
Region_name	string…

Indicatemarkadapt

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

```
Indicatemarkadapt Region_name Using Indicator Marker [ When When-expression ]
```

Region_name	string
Indicator	string
Marker	string

Involve

Specifiy a physics participant to a coupled problem solved using matrix-free nonlinear.

```
Involve Name
```

Name	string

Mark

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

```
Mark Region_name… Using Marker_name… [ When When-expression ]
```

Region_name	string…
Marker_name	string…

Markadapt

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

```
Markadapt Region_name Using Marker [ When When-expression ]
```

Region_name	string
Marker	string

Output

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

```
Output Name [ When When-expression ]
```

Name	string

Transfer

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.

```
Transfer Name [ When When-expression ]
```

Name	string

8.6 Subcycle

This block is used to wrap a subcycle time loop.

```
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
```

Adapt

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

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

Region_name	string…
Field_name	string…

Advance

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

```
Advance Name… [ When When-expression ]
```

Name	string…

Compute Indicator On

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

```
Compute Indicator On Region_name… Using Indicator_name… [ When When-expression ]
```

Region_name	string…
Indicator_name	string…

Event

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.

```
Event Name… [ When When-expression ]
```

Name	string…

Execute Postprocessor Group

Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

```
Execute Postprocessor Group Group_name… On Region_name… [ When When-expression ]
```

Group_name	string…
Region_name	string…

Indicatemarkadapt

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

```
Indicatemarkadapt Region_name Using Indicator Marker [ When When-expression ]
```

Region_name	string
Indicator	string
Marker	string

Involve

Specifiy a physics participant to a coupled problem solved using matrix-free nonlinear.

```
Involve Name
```

Name	string

Mark

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

```
Mark Region_name… Using Marker_name… [ When When-expression ]
```

Region_name	string…
Marker_name	string…

Markadapt

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

```
Markadapt Region_name Using Marker [ When When-expression ]
```

Region_name	string
Marker	string

Output

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

```
Output Name [ When When-expression ]
```

Name	string

Transfer

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.

```
Transfer Name [ When When-expression ]
```

Name	string

8.7 Sequential

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.

```
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
```

Adapt

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

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

Region_name	string…
Field_name	string…

Advance

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

```
Advance Name… [ When When-expression ]
```

Name	string…

Compute Indicator On

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

```
Compute Indicator On Region_name… Using Indicator_name… [ When When-expression ]
```

Region_name	string…
Indicator_name	string…

Event

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.

```
Event Name… [ When When-expression ]
```

Name	string…

Execute Postprocessor Group

Used within a Solver Control block to cause the group named group_name to be executed on region region_name.

```
Execute Postprocessor Group Group_name… On Region_name… [ When When-expression ]
```

Group_name	string…
Region_name	string…

Indicatemarkadapt

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

```
Indicatemarkadapt Region_name Using Indicator Marker [ When When-expression ]
```

Region_name	string
Indicator	string
Marker	string

Involve

Specifiy a physics participant to a coupled problem solved using matrix-free nonlinear.

```
Involve Name
```

Name	string

Mark

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

```
Mark Region_name… Using Marker_name… [ When When-expression ]
```

Region_name	string…
Marker_name	string…

Markadapt

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

```
Markadapt Region_name Using Marker [ When When-expression ]
```

Region_name	string
Marker	string

Output

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

```
Output Name [ When When-expression ]
```

Name	string

Transfer

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.

```
Transfer Name [ When When-expression ]
```

Name	string

8.8 Initialize

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.

```
Begin Initialize Name
   Advance Name… [ When When-expression ]
   Event Name… [ When When-expression ]
   Involve Name
   Transfer Name [ When When-expression ]
End
```

Advance

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

```
Advance Name… [ When When-expression ]
```

Name	string…

Event

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.

```
Event Name… [ When When-expression ]
```

Name	string…

Involve

Specifiy a physics participant to a coupled problem solved using matrix-free nonlinear.

```
Involve Name
```

Name	string

Transfer

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.

```
Transfer Name [ When When-expression ]
```

Name	string

8.9 Parameters For

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.

```
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
End
```

Converged When

Set the convergence expression.

```
Converged When Convergence-expression
```

Convergence-expression	(expression)

Incremental Number Of Steps

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

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

Number	integer

Initial Deltat

Assign an initial delta T

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

Number	real

Number Of Adaptivity Steps

The number steps to run the time or nonlinear loop

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

Number	integer

Number Of Steps

The number steps to run the time for nonlinear loop

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

Number	integer

Reinitialize Transient

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

```
Reinitialize Transient
```

Start Time

Assign a start time.

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

Number	real

Termination Time

Assign a final time to stop

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

Number	real

Time Step Quantum

Set the time stepping quantum time for SNAP style stepping.

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

TimeStepQuantum	real

Time Step Style

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.

```
Time Step Style TimeStepStyle…
```

TimeStepStyle	{clip|noclip|nosnap|snap}

Total Change In Time

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

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

Number	real

Chapter 9

Input and Output Commands

9.1 Finite Element Model

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

```
Begin Finite Element Model Label
   Alias DatabaseName As InternalName
   Component Separator Character Option Separator
   Create GroupType NewSurfaceName Add SurfaceName…
   Coordinate System {=|are|is} CoordinateSystem
   Database Name {=|are|is} StreamName
   Database Type {=|are|is} DatabaseTypes
   Decomposition Method {=|are|is} Method
   Global Id Mapping Backward Compatibility Option1 Option2
   Omit Block BlockList…
   Omit Volume VolumeList…
End
```

Alias

Name the database entity "DatabaseName" as "InternalName"

```
Alias DatabaseName As InternalName
```

DatabaseName	string
InternalName	string

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

Component Separator Character

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.

```
Component Separator Character Option Separator
```

Separator	string

Create

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.

```
Create GroupType NewSurfaceName Add SurfaceName…
```

NewSurfaceName	string
SurfaceName	string…

Coordinate System

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

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

CoordinateSystem	{axisymmetric | barycentric | cartesian | cyclidic | cylindrical | polar | quadriplanar | skew | spherical | toroidal | trilinear}

Database Name

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.

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

StreamName	string

Database Type

The database type/format used for the mesh.

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

DatabaseTypes	{dof | dof_exodus | exodus | exodusii | generated | genesis | parallel_exodus | paraview | paraview_catalyst | xdmf}

Decomposition Method

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

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

Global Id Mapping Backward Compatibility

(Unsupported, do not use)

```
Global Id Mapping Backward Compatibility Option1 Option2
```

Omit Block

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

```
Omit Block BlockList…
```

BlockList	string…

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.

Omit Volume

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

```
Omit Volume VolumeList…
```

VolumeList	string…

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.

9.2 Results Output

Describes the location and type of the output stream used for outputting results for the enclosing 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… ]
   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
   Timestep Adjustment Interval {=|are|is} Nsteps
   Title
   Use Output Scheduler Timer_name
End
```

Additional Steps

Additional simulation steps when output should occur.

```
Additional Steps {=|are|is} List_of_steps…
```

List_of_steps	integer…

Additional Times

Additional simulation times when output should occur.

```
Additional Times {=|are|is} List_of_times…
```

List_of_times	real…

At Step

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.

```
At Step n Option {=|are|is} m
```

n	integer
m	integer

At Time

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.

```
At Time Dt1 Option {=|are|is} Dt2
```

Dt1	real
Dt2	real

Component Separator Character

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.

```
Component Separator Character {=|are|is} Separator
```

Separator	string

Database Name

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.

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

StreamName	string

Database Type

The database type/format to be used for the output results.

```
Database Type {=|are|is} DatabaseType
```

Edge

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.

```
Edge [ VariableList… ]
```

Edge Variables

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.

```
Edge Variables {=|are|is} [ VariableList… ]
```

Element

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"

```
Element [ VariableList… ]
```

Element Variables

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"

```
Element Variables {=|are|is} [ VariableList… ]
```

Exclude

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.

```
Exclude {=|are|is} [ ElementBlockList… ]
```

Exists

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.

```
Exists Option1 Option2
```

Face

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.

```
Face [ VariableList… ]
```

Face Variables

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.

```
Face Variables {=|are|is} [ VariableList… ]
```

Global

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.

```
Global [ Variables… ]
```

Global Variables

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.

```
Global Variables {=|are|is} [ Variables… ]
```

Include

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.

```
Include {=|are|is} [ ElementBlockList… ]
```

Nodal

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.

```
Nodal [ VariableList… ]
```

Nodal Variables

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.

```
Nodal Variables {=|are|is} [ VariableList… ]
```

Node

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.

```
Node [ VariableList… ]
```

Node Variables

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.

```
Node Variables {=|are|is} [ VariableList… ]
```

Nodeset

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.

```
Nodeset [ VariableList… ]
```

Nodeset Variables

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.

```
Nodeset Variables {=|are|is} [ VariableList… ]
```

Output Mesh

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.

```
Output Mesh {=|are|is} OutputMesh
```

OutputMesh	{block surface|exposed surface|refined|unrefined}

Output On Signal

When the specified signal is raised, the output stream associated with this block will be output.

```
Output On Signal {=|are|is} Signals
```

Signals	{sigabrt|sigalrm|sigfpe|sighup|sigill|sigint|sigkill|sigpipe|sigquit|sigsegv|sigterm|sigusr1|sigusr2}

Overwrite

(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.

```
Overwrite Option1 Option2
```

Property

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.

```
Property PropertyName {=|are|is} PropertyValue
```

PropertyName	string
PropertyValue	string

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)
```

Sideset

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.

```
Sideset [ VariableList… ]
```

Sideset Variables

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.

```
Sideset Variables {=|are|is} [ VariableList… ]
```

Start Time

Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

```
Start Time {=|are|is} Start_time
```

Start_time	real

Surface

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.

```
Surface [ VariableList… ]
```

Surface Variables

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.

```
Surface Variables {=|are|is} [ VariableList… ]
```

Synchronize Output

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.

```
Synchronize Output
```

Termination Time

Specify the time to stop outputting results from this output request block.

```
Termination Time {=|are|is} Final_time
```

Final_time	real

Timestep Adjustment Interval

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'.

```
Timestep Adjustment Interval {=|are|is} Nsteps
```

Nsteps	integer

Title

Specify the title to be used for this specific output block.

```
Title
```

Use Output Scheduler

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

```
Use Output Scheduler Timer_name
```

Timer_name	string

9.3 Restart Data

Describes the data required to output and input restart data for the enclosing 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
   Component Separator Character Option Separator
   Cycle Count {=|are|is} Count
   Database Name {=|are|is} StreamName
   Database Type {=|are|is} DatabaseTypes
   Debug Dump
   Dump All
```

```
   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} Option
   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
```

Additional Steps

Additional simulation steps when output should occur.

```
Additional Steps {=|are|is} List_of_steps…
```

List_of_steps	integer…

Additional Times

Additional simulation times when output should occur.

```
Additional Times {=|are|is} List_of_times…
```

List_of_times	real…

At Step

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.

```
At Step n Option {=|are|is} m
```

n	integer
m	integer

At Time

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.

```
At Time Dt1 Option {=|are|is} Dt2
```

Dt1	real
Dt2	real

Component Separator Character

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.

```
Component Separator Character Option Separator
```

Separator	string

Cycle Count

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.

```
Cycle Count {=|are|is} Count
```

Count	integer

Database Name

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.

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

StreamName	string

Database Type

The database type/format used for the restart file.

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

DatabaseTypes	{dof | dof_exodus | exodus | exodusii | generated | genesis | parallel_exodus | paraview | paraview_catalyst | xdmf}

Debug Dump

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.

```
Debug Dump
```

Dump All

Specify that the restart system should treat all variables as needed for restart whether they are persistent, temporary, or constant. Used only for debugging restart.

```
Dump All
```

Decomposition Method

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

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

Exists

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.

```
Exists Option1 Option2
```

File Cycle Count

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.

```
File Cycle Count {=|are|is} Count
```

Count	integer

Input Database Name

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.

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

StreamName	string

Optional

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.

```
Optional
```

Output Database Name

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.

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

StreamName	string

Output On Signal

When the specified signal is raised, the output stream associated with this block will be output.

```
Output On Signal {= | are | is} Signals
```

Signals	{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}

Overlay Count

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.

```
Overlay Count {= | are | is} Count
```

Count	integer

Overwrite

(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.

```
Overwrite Option1 Option2
```

Property

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.

```
Property PropertyName {=|are|is} PropertyValue
```

PropertyName	string
PropertyValue	string

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
```

Restart

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.

```
Restart {=|are|is} Option
```

Restart Time

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.

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

Time	real

Start Time

Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

```
Start Time {=|are|is} Start_time
```

Start_time	real

Synchronize Output

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.

```
Synchronize Output
```

Termination Time

Specify the time to stop outputting results from this output request block.

```
Termination Time {=|are|is} Final_time
```

Final_time	real

Timestep Adjustment Interval

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'.

```
Timestep Adjustment Interval {=|are|is} Nsteps
```

Nsteps	integer

Use Output Scheduler

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

```
Use Output Scheduler Timer_name
```

Timer_name	string

9.4 Heartbeat

Describes the location and type of the output stream used for outputting the heartbeat information for the enclosing 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
   Edge [ VariableList… ]
   Element [ VariableList… ]
   Exists Option1 Option2
   Face [ VariableList… ]
   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
```

Additional Steps

Additional simulation steps when output should occur.

```
Additional Steps {=|are|is} List_of_steps…
```

List_of_steps	integer…

Additional Times

Additional simulation times when output should occur.

```
Additional Times {=|are|is} List_of_times…
```

List_of_times	real…

Append

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.

```
Append {=|are|is} Option
```

At Step

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.

```
At Step n Option {=|are|is} m
```

n	integer
m	integer

At Time

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.

```
At Time Dt1 Option {=|are|is} Dt2
```

Dt1	real
Dt2	real

Edge

Define the edge variables that should be written to the heartbeat database. The syntax is: "edge internal_name at edge id as DBname" or "edge 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.

```
Edge [ VariableList… ]
```

Element

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.

```
Element [ VariableList… ]
```

Exists

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.

```
Exists Option1 Option2
```

Face

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.

```
Face [ VariableList… ]
```

Format

The stream type/format to be used for the output results.The only two options at this time are 'Default' which is the normal Sierra heartbeat format; and 'SpyHis' which mimics the CTH Spyhis history output format.

```
Format {=|are|is} StreamTypes
```

StreamTypes	{default|spyhis}

Global

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.

```
Global [ Variables… ]
```

Labels

Specifies whether labels will be displayed or just the value of the variable. Labels will be shown if this line is not present.

```
Labels {=|are|is} Option
```

Legend

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).

```
Legend {=|are|is} Option
```

Monitor

Specifies whether a line will be written to the heartbeat stream when either the results, history, and/or restart data are output.

```
Monitor Equals Option
```

Nodal

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.

```
Nodal [ VariableList… ]
```

Node

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.

```
Node [ VariableList… ]
```

Nodeset

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.

```
Nodeset [ VariableList… ]
```

Output On Signal

When the specified signal is raised, the output stream associated with this block will be output.

```
Output On Signal {=|are|is} Signals
```

Signals	{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}

Precision

The precsion to be used for the output of real variables.

```
Precision {=|are|is} Precision
```

Precision	integer

Start Time

Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

```
Start Time {=|are|is} Start_time
```

Start_time	real

Stream Name

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.

```
Stream Name {=|are|is} OutputFilename
```

OutputFilename	string

Synchronize Output

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.

```
Synchronize Output
```

Termination Time

Specify the time to stop outputting results from this output request block.

```
Termination Time {=|are|is} Final_time
```

Final_time	real

Timestamp Format

The format to be used for the timestamp. See 'man strftime' for more information.

```
Timestamp Format
```

Timestep Adjustment Interval

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'.

```
Timestep Adjustment Interval {=|are|is} Nsteps
```

Nsteps	integer

Use Output Scheduler

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

```
Use Output Scheduler Timer_name
```

Timer_name	string

Variable

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.

```
Variable {=|are|is} Option [ Variable_list… ]
```

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.

9.5 History Output

Describes the location and type of the output stream used for outputting history for the enclosing 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
   Debug
   Edge [ VariableList… ]
   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
```

Additional Steps

Additional simulation steps when output should occur.

```
Additional Steps {=|are|is} List_of_steps…
```

List_of_steps	integer…

Additional Times

Additional simulation times when output should occur.

```
Additional Times {=|are|is} List_of_times…
```

List_of_times	real…

At Step

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.

```
At Step n Option {=|are|is} m
```

n	integer
m	integer

At Time

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.

```
At Time Dt1 Option {=|are|is} Dt2
```

Dt1	real
Dt2	real

Database Name

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.

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

StreamName	string

Database Type

The database type/format to be used for the output history.

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

DatabaseTypes	{dof | dof_exodus | exodus | exodusii | generated | genesis | parallel_exodus | paraview | paraview_catalyst | xdmf}

Debug

Turn on debugging output.

```
Debug
```

Edge

Define the edge variables that should be written to the history database. The syntax is: "edge internal_name at edge id as DBname" or "edge 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.

```
Edge [ VariableList… ]
```

Element

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.

```
Element [ VariableList… ]
```

Exists

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.

```
Exists Option1 Option2
```

Face

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.

```
Face [ VariableList… ]
```

Global

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.

```
Global [ Variables… ]
```

Nodal

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.

```
Nodal [ VariableList… ]
```

Node

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.

```
Node [ VariableList… ]
```

Nodeset

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.

```
Nodeset [ VariableList… ]
```

Output On Signal

When the specified signal is raised, the output stream associated with this block will be output.

```
Output On Signal {=|are|is} Signals
```

Signals	{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}

Overwrite

(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.

```
Overwrite Option1 Option2
```

Property

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)

```
Property PropertyName {=|are|is} PropertyValue
```

PropertyName	string
PropertyValue	string

Start Time

Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

```
Start Time {=|are|is} Start_time
```

Start_time	real

Synchronize Output

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.

```
Synchronize Output
```

Termination Time

Specify the time to stop outputting results from this output request block.

```
Termination Time {=|are|is} Final_time
```

Final_time	real

Timestep Adjustment Interval

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'.

```
Timestep Adjustment Interval {=|are|is} Nsteps
```

Nsteps	integer

Title

Specify the title to be used for this specific output block.

```
Title
```

Use Output Scheduler

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

```
Use Output Scheduler Timer_name
```

Timer_name	string

Variable

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.

```
Variable {=|are|is} Option [ Variable_list… ]
```

Appendix

Example 1 Interpolating a String Function, and Evaluation

```
Begin Sierra Encore

  Begin Postprocessor Output Control pp_out
    Write To File input1_out.txt
    Output To Console
  End

  Begin Finite Element Model lshaped_mesh
     Database Name = mesh/3d_lshaped.e
  End

  Begin String Function sfunc
    Value Is "x-2" "(t*y)+2" "z+3"
    Gradient Is "1" "0" "0" "0" "t" "0" "0" "0" "1"
    Time Derivative Is "0" "y" "0"
  End

  Begin Encore Procedure encore_proc
    Begin Encore Region encore_region
      Use Finite Element Model lshaped_mesh Model Coordinates Are model_coordinates

      Evaluate Function sfunc At -1 -1 -1 1
      Evaluate Gradient Of sfunc At -1 -1 -1 1
      Evaluate Time Derivative Of sfunc At -1 -1 -1 1

      Interpolate Function Value Of sfunc Into Nodal Field func_values
      Interpolate Function Gradient Of sfunc Into Nodal Field func_gradient
      Interpolate Function Time Derivative Of sfunc Into Nodal Field func_dot

      Begin Results Output Label encore_adapt_nodeset_results
        Database Name = input1_out.e
        Database Type = exodusII
        At Step 0 Increment = 1
        Nodal Variables = func_values
        Nodal Variables = func_gradient
        Nodal Variables = func_dot
      End Results Output
    End
```

```
  End Encore Procedure
End Sierra
```

Example 2 String Functions, a Difference Function, and Norms

```
Begin Sierra Encore

  Begin Postprocessor Output Control pp_out
    Write To File input3_out.txt
    Output To Console
  End

  Begin Finite Element Model lshaped_mesh
     Database Name = mesh/3d_lshaped.e
  End

  Begin String Function sfunc
     Value Is "x-2" "(t*y)+2" "z+3"
     Gradient Is "1" "0" "0" "0" "t" "0" "0" "0" "1"
     Time Derivative Is "0" "y" "0"
  End

  Begin String Function other_sfunc
     Value Is "(x*x)-2" "(t*y)+2" "z+3"
     Gradient Is "2*x" "0" "0" "0" "t" "0" "0" "0" "1"
     Time Derivative Is "0" "y" "0"
  End

  Begin Difference Function diff_func
     Difference Is sfunc - other_sfunc
  End

  Begin Encore Procedure encore_proc
    Begin Encore Region encore_region
      Use Finite Element Model lshaped_mesh Model Coordinates Are model_coordinates

      Compute Norm L2 Of sfunc
      Compute Norm H1 Of sfunc Store In sfunc_h1
      Compute Norm LInfinity Of sfunc
      Compute Norm Nodal LInfinity Of sfunc
      Compute Difference L2 Of sfunc other_sfunc Store In l2_diff
      Compute Norm L2 Of diff_func

      Begin Results Output Label encore_adapt_nodeset_results
        Database Name = input3_out.e
        Database Type = exodusII
        At Step 0 Increment = 1
        Element Variables = sfunc_h1
        Element Variables = l2_diff
      End Results Output
    End
  End Encore Procedure
End Sierra
```

Example 3 Field Function Import, Evaluate at Point, and Norm

```
Begin Sierra Encore

    Begin Postprocessor Output Control pp_out
        Output To Console
    End

    Begin Finite Element Model conv2d
        Database Name = mesh/conv2d.e
    End

    Begin Field Function ffunc
        Use Nodal Field temperature # use nodal coefficients from imported field (below)
    End

    Begin Encore Procedure my_procedure
        Begin Encore Region my_region
            Use Finite Element Model conv2d Model Coordinates Are model_coordinates
            Import Field TND as Nodal Field temperature # import from mesh file conv2d
            Evaluate Function ffunc At .5 .5
            Compute Norm L2 Of ffunc
     End Encore Region
  End Encore Procedure
End Sierra
```

Example 4 User Function, Evaluate Gradient, and Evaluate Time Derivative

```
Begin Sierra Encore

  Title User Function Example

  Begin Postprocessor Output Control pp_out
    Output To Console
    Write To File input1_out.txt
  End

  Begin Finite Element Model conv2d
     Database Name = mesh/conv2d.e
  End Finite Element Model

  Begin User Function ufunc
     Load From File ./userFunc.so Using Function registerUserFunc
     Parameter specific_heat = 4
  End

  Begin Encore Procedure encore_ffunc_proc
    Begin Encore Region encore_ffunc_reg

      Use Finite Element Model conv2d Model Coordinates Are model_coordinates

      Evaluate Function ufunc At 4 1 0 2
      Evaluate Function ufunc At 6 1 0 2
      Evaluate Gradient Of ufunc At 1 2 0 3
      Evaluate Time Derivative Of ufunc At 2 3 0 1

      Interpolate Function Value Of ufunc Into Nodal Field uvalue

      Begin Results Output Label encore_adapt_nodeset_results
        Database Name = input1_out.e
        Database Type = exodusII
        At Step 0 Increment = 1
        Nodal Variables = uvalue
      End Results Output
    End Encore Region
  End Encore Procedure
End Sierra
```

Source Code for User Function, the file: `userFunc.C`

```
#include<iostream>

#include<functions/Encr_UserFunctionParser.h>
#include<functions/Encr_UserFunction.h>
#include<functions/Encr_FunctionFactory.h>

#include<framewk/Fmwk_Procedure.h>
```

```
#include<framewk/Fmwk_Region.h>
#include<framewk/Fmwk_Domain.h>

namespace sierra
{
namespace Encr
{
// Forward declaration just so we can put the registration function at the top.
class SomeFunc;

// This is the registration function... you will actually be referencing this
// in the input file so give it a good name.  What it does is register your
// function object with the FunctionFactory so it can be built.
// Note that a seperate one of these is necessary for every user function
// defined in this file.
extern "C" void registerUserFunc()
{
  UserFunctionParser::register_user_function<SomeFunc>();
}

/*
  This is a piecewise discontinuous function.
  It's (c*x^2 + y*t) when x < 4.5 and (c*x^3 - y*t) when x >=4.5

  It inherits off of UserFunction.  You don't have to override all of these virtuals.
  For instance, if all you are interested in is L2 Norm then you don't need gradient or
  time derivative.
*/
class SomeFunc : public UserFunction
{
public:

  //Constructor... always has to be here
  SomeFunc(const Fmwk::Region * reg, const Fmwk::Parameters * params)
    :UserFunction(reg,params)
  {
    //This needs to match the dimension of the return value from value()
    dimension=1;
  }

  //This is the "Value" of the function or f(x)... "point" will be sized appropriately
  //for the spatial dimension and "time" is the current time.
  virtual std::vector<double> value(const std::vector<double>& point, const double& time)
  {
    std::vector<double> value(1);

    double c = getParam("specific_heat");

    double x = point[0];
    double y = point[1];

    if(x < 4.5)
```

```
      value[0] = c*x*x + y*time + 3000;
    else
      value[0] = c*x*x*x - y*time;

    return value;
  }

  //This is the "Gradient" of the function
  virtual std::vector<double> gradient(const std::vector<double>& point,
                                       const double& time)
  {
    std::vector<double> gradient(2);

    double c = getParam("specific_heat");

    double x = point[0];

    if(x < 4.5)
    {
      gradient[0]=2*c*x;
      gradient[1]=time;
    }
    else
    {
      gradient[0]=3*c*x*x;
      gradient[1]=-time;
    }

    return gradient;
  }

  //And finally the time derivative.
  virtual std::vector<double> dot(const std::vector<double>& point, const double& time)
  {
    std::vector<double> value(1);

    double x = point[0];
    double y = point[1];

    if(x < 4.5)
      value[0]=y;
    else
      value[0]=-y;

    return value;
  }
};  //Don't forget that closing semicolon....

} // namespace sierra
} // namespace encr
```

Example 5 Norm Postprocessor, Compute Difference

```
Begin Sierra Encore

  Title Unit test for the l2 norm

  Begin Postprocessor Output Control pp_out
    Output To Console
    Write To File encoreinfo.txt
  End

  Begin Finite Element Model rectangular_prism
     Database Name = rectangular_prism.e
  End Finite Element Model

  Begin String Function sfunc
     Value Is "x"
     Gradient Is "1" "0" "0"
  End

  Begin String Function sfunc2
     Value Is "x+1"
     Gradient Is "1" "0" "0"
  End

  Begin String Function sfunc3
     Value Is "2*x"
     Gradient Is "2" "0" "0"
  End

  Begin Field Function efunc
     Use Element Field sfunc_interp
  End

  Begin Norm Postprocessor npp
     Use Function sfunc
     Compute Norms L2 H1 L1
  End

  Begin Norm Postprocessor npp_diff
     Use Function sfunc
     Subtract Function sfunc2
     Compute Norms L2 H1 L1
  End

  Begin Norm Postprocessor npp_block_2
     Use Function sfunc
     Compute Norm L2
     Volumes block_2
     Store In l2_block2
  End

  Begin Norm Postprocessor npp_block_1
```

```
   Use Function sfunc
   Compute Norm L2
   Omit Volumes block_2
   Store In l2_block1
End

Begin Norm Postprocessor npp_surface_1
   Use Function sfunc
   Compute Norm L2
   Surfaces surface_1
   #Store In l2_surface1
End

Begin Norm Postprocessor npp_h1_surface_2
   Use Function sfunc
   Compute Norm H1 Restriction
   Surfaces surface_2
   #Store In h1_restrict_surface1
End

Begin Norm Postprocessor npp_h1_diff_surface_2
   Use Function sfunc
   Subtract Function sfunc3
   Compute Norm Relative H1 Restriction
   Surfaces surface_2
   #Store In h1_restrict_surface1
End

Begin Norm Postprocessor npp_l2_elem_surface_1
   Use Function efunc
   Compute Norm L2
   Surfaces surface_1
End

Begin Encore Procedure encore_ffunc_proc
  Begin Encore Region encore_ffunc_reg

    Use Finite Element Model rectangular_prism Model Coordinates Are model_coordinates

    Interpolate Function Value Of sfunc Into Element Field sfunc_interp

    Compute Norm L2 Of sfunc
    Compute Norm H1 Of sfunc
    Compute Norm L1 Of sfunc

    Evaluate Postprocessor npp

    Compute Difference L2 Of sfunc sfunc2
    Compute Difference H1 Of sfunc sfunc2
    Compute Difference L1 Of sfunc sfunc2

    Evaluate Postprocessor npp_diff
```

```
      #These have been analytically verified using Maple
      Evaluate Postprocessor npp_block_2
      Evaluate Postprocessor npp_block_1
      Evaluate Postprocessor npp_surface_1

      # value checked for correctness
      Evaluate Postprocessor npp_h1_surface_2
      Evaluate Postprocessor npp_h1_diff_surface_2

      Evaluate Postprocessor npp_l2_elem_surface_1

      Begin Results Output Label encore_adapt_nodeset_results
        Database Name = rectangular_prism_out.e
        Database Type = exodusII
        Title Adapted 3D L-Shaped Domain
        At Step 0 Increment = 1
        Element Variables = l2_block2
        Element Variables = l2_block1
        #Element Variables = l2_surface1
        #Element Variables = h1_restrict_surface1
      End Results Output

    End Encore Region
  End Encore Procedure
End Sierra
```

Example 6 Region Level Norms

```
Begin Sierra Encore

  Title Unit test for relative error

  Begin Postprocessor Output Control pp_out
    Output To Console
    Write To File encoreinfo.txt
  End

  Begin Finite Element Model square_four_quad4
     Database Name = square_four_quad4.e
  End

  Begin String Function sfunc2
    Value is "sin(x+y)"
    Gradient is "cos(x+y)" "cos(x+y)"
  End

  Begin Field Function ffunc
     Use Nodal Field temperature
  End

  Begin Encore Procedure encore_ffunc_proc
    Begin Encore Region region_square_four_quad4

      Use Finite Element Model square_four_quad4 Model Coordinates Are model_coordinates

      Interpolate Function Value Of sfunc2 Into Nodal Field temperature

      Compute Norm       L2          of sfunc2
      Compute Difference L2          of sfunc2 ffunc Store In l2_diff
      Compute Difference Relative L2 of sfunc2 ffunc Store In l2_rel_diff

      Compute Norm       L1          of sfunc2
      Compute Difference L1          of sfunc2 ffunc Store In l1_diff
      Compute Difference Relative L1 of sfunc2 ffunc Store In l1_rel_diff

      Compute Norm       H1          of sfunc2
      Compute Difference H1          of sfunc2 ffunc Store In h1_diff
      Compute Difference Relative H1 of sfunc2 ffunc Store In h1_rel_diff

      Compute Norm       LInfinity          of sfunc2
      Compute Difference LInfinity          of sfunc2 ffunc Store In linf_diff
      Compute Difference Relative LInfinity of sfunc2 ffunc Store In linf_rel_diff

      Compute Norm       Nodal LInfinity          of sfunc2
      Compute Difference Nodal LInfinity          of sfunc2 ffunc Store In nodal_linf_diff
      Compute Difference Relative Nodal LInfinity of sfunc2 ffunc Store In nodal_linf_rdiff

      Begin Results Output Label encore_recover_gradient
        Database Name = square_four_quad4_out.e
```

```
        Database Type = exodusII
        Title Adapted 2D Convection
        At Step 1 Increment = 1
        Nodal Variables = temperature

        Element Variables = l2_diff
        Element Variables = l2_rel_diff
        Element Variables = l1_diff
        Element Variables = l1_rel_diff
        Element Variables = h1_diff
        Element Variables = h1_rel_diff
        Element Variables = linf_diff
        Element Variables = linf_rel_diff
        Nodal Variables = nodal_linf_diff
        Nodal Variables = nodal_linf_rdiff
      End Results Output

    End Encore Region

  End Encore Procedure
End Sierra
```

Example 7 User Postprocessor

```
Begin Sierra Encore

  Title Test for Nodal Integral Postprocessor

  Begin Postprocessor Output Control pp_out
    Write To File input2_out.txt
  End

  Begin Finite Element Model box_out
    Database Name = mesh/box_out.e
  End Finite Element Model

  Begin Field Function t_func
    Use Nodal Field t
  End

  Begin Field Function t_integral_func
    Use Nodal Field t_integral
  End

  Begin Field Function user_eq_func
    Use Nodal Field user_eq_cure_field
  End

  Begin User Postprocessor user_nodal_integral
    Load From File ./nodalIntegral.so Using Function registerNodalIntegral
    Use Function t_func
    Store In t_integral
  End

  Begin Encore Procedure encore_proc

    Begin Solution Control Description
      Use System main
      Begin System main
        Begin Transient encore_trans
          Advance encore_reg
        End
        Simulation Start Time = 0
        Simulation Termination Time = 100.0
        Simulation max global iterations = 1000
      End
    End

    Begin Encore Region encore_reg
      Use Finite Element Model box_out Model Coordinates Are model_coordinates

      Import Field TND as Nodal Field t

      Evaluate Postprocessor user_nodal_integral
```

```
      Evaluate Function t_func          At 0.5 0.5 0
      Evaluate Function t_integral_func At 0.5 0.5 0

      Begin Results Output Label encore_equivalent_cure
        Database Name = input2_out.e
        Database Type = exodusII
        Title Adapted 2D Convection Postprocessed
        At Step 1 Increment = 1
        Nodal Variables = t
        Nodal Variables = t_integral
      End Results Output

    End Encore Region
  End Encore Procedure
End Sierra
```

Source Code for User Postprocessor, the file: `nodalIntegral.C`

```
#include<postprocessors/Encr_UserPostprocessorParser.h>
#include<postprocessors/Encr_NodalUserPostprocessor.h>

#include<framewk/Fmwk_NodalBLAS.h>

namespace sierra
{

namespace Encr
{

// Forward declaration just so we can put the registration function at the top.
class NodalIntegral;

// This is the registration function... you will actually be referencing this
// in the input file so give it a good name.  What it does is register your
// postprocessor object with the PostprocessorFactory so it can be built.
extern "C" void registerNodalIntegral()
{
  UserPostprocessorParser::register_user_postprocessor<NodalIntegral>();
}

/*
  This Postprocessor calculates a time integral value at each node.
*/
class NodalIntegral : public NodalUserPostprocessor
{
public:

  //Constructor... always has to be here
  NodalIntegral(const Fmwk::Region * reg, const Fmwk::Parameters * params)
    :NodalUserPostprocessor(reg,params)
  {
    //Setting "should_output" to false means that this postprocessor won't output anything
```

```
    //into the postprocessor output table.
    //If you do want something to come out in the table, set this to true and return some
    //number from postLoop()
    should_output = false;
  }

protected:
  //This is what will actually get excecuted at each node, with the node being passed in.
  virtual void kernel(const Fmwk::MeshObj* obj);

  //Preloop runs before the nodal loop starts, so do preprocessing here.
  virtual void preLoop();

private:
};  //Don't forget that closing semicolon....

void
NodalIntegral::preLoop()
{
  //If the field has more than one state, copy it to STATE_NEW
  //This is specific to the way this PP works.
  Fmwk::Field * f = const_cast<Fmwk::Field*>(field.field());
  if (f->state_length()>1)
    Fmwk_nodal_copy(*const_cast<Fmwk::Region*>(region),
    Fmwk::FieldRef(f,Fmwk::STATE_OLD),
    Fmwk::FieldRef(f,Fmwk::STATE_NEW));

}

void
NodalIntegral::kernel(const Fmwk::MeshObj* obj)
{
  //"field" is the FieldRef that is set by the "Use Field" command
  Real * v_int = obj->data(field);

  const double t0 = oldTime();
  const double t1 = curTime();

  // scalar values are the first elements in the vector returned by the eval
  double v0 = function->valueNode(obj, t0)[0];
  double v1 = function->valueNode(obj, t1)[0];

  // sum in the trapezoid contribution for this time step into the integral
  *v_int += (t1 - t0) * (v0 + v1) / 2.0;
}

} // namespace sierra
} // namespace encr
```

Example 8 Transfer on a Subinterval of Time

```
Begin Sierra Encore

  Title Tests Transfer Element Field Fine to Coarse

  Begin Finite Element Model solution_mesh
     Database Name = t_time_rythmos_verify_medium.e
  End

  Begin Finite Element Model transfer_mesh
     Database Name = block_hex8_medium.e
  End

  Begin Postprocessor Output Control pp_out
    Output To Console
    Write To File encoreinfo.txt
  End

  Begin Encore Procedure encore_proc
    Begin Solution Control Description
      Use System main
      Begin System main
        Begin Transient encore_trans
          Advance I_region
          Transfer I_to_O
          Advance O_region
        End
        Simulation Start Time = 0.25
        Simulation Termination Time = 0.75
        Simulation Max Global Iterations = 100
      End
    End

    Begin Transfer I_to_O
        Copy Volume Nodes From I_region To O_region
        Send Field T State New To T_transferred State New
    End

    Begin Encore Region I_region
      Use Finite Element Model solution_mesh Model Coordinates Are model_coordinates

      Output Number of Nodes

      #Process Initial Condition

      Import Field T As Nodal Field T
    End Encore Region

    Begin Encore Region O_region
      Use Finite Element Model transfer_mesh Model Coordinates Are model_coordinates

      Create Nodal Field T_transferred Of Type REAL
```

```
      Disable Compute Timestep

      Output Number of Nodes

      Begin Results Output Label O_region
        Database Name = transfer_sub_time_interval_out.e
        Database Type = exodusII
        Title Adapted 3D L-Shaped Domain
        At Step 0 Increment = 1
        Nodal Variables = T_transferred As T
      End Results Output
    End Encore Region

  End Encore Procedure
End Sierra
```

Index

Distribution

Sandia Internal Distribution

2	MS 0828	1544	B. Carnes
2	MS 0828	1544	K. D. Copps
1	MS 0828	1544	W. R. Witkowski
1	MS 0807	9326	A. M. Agelastos
1	MS 0807	9326	R. P. Shaw
1	MS 0823	9326	T. M. Hensley
1	MS 0845	9326	J. D. Miller
1	MS 0899	9532	RIM-Reports Management (*electronic copy*)


Sandia National Laboratories