**SAND REPORT**
SAND2003-1089
Unlimited Release
Printed April 2003

# Presto User's Guide Version 1.05

J. Richard Koteras and Arne S. Gullerud

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

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy 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.doe.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 order: http://www.ntis.gov/ordering.htm

SAND2003-1089
Unlimited Release
Printed April 2003

# Presto User's Guide Version 1.05

J. Richard Koteras and Arne S. Gullerud
Computational Solid Mechanics/Structural Dynamics
Engineering Sciences Center
Sandia National Laboratories
Box 5800
Albuquerque, NM 87195-0847

**Abstract**

Presto is a Lagrangian, three-dimensional explicit, transient dynamics code for the analysis of solids subjected to large, suddenly applied loads. Presto is designed for problems with large deformations, nonlinear material behavior, and contact. There is a versatile element library incorporating both continuum and structural elements. The code is designed for a parallel computing environment. This document describes the input for the code that gives users access to all of the current functionality in the code. Presto is built in an environment that allows it to be coupled with other engineering analysis codes. The input structure for the code, which uses a concept called scope, reflects the fact that Presto can be used in a coupled environment. This guide describes the scope concept and the input from the outermost to the innermost input scopes. Within a given scope, the descriptions of input commands are grouped based on code functionality. For example, all material input command lines are described in a section of the user's guide for all of the material models in the code.

# Acknowledgments

The authors would like to thank Joel Lash and Nathan Crane for reviewing this document and offering many helpful suggestions for improving its clarity. The authors would also like to thank Ken Gwinn, William Scherzinger, and Henry Duong for comments, suggestions, and assistance in improving various parts of the user's guide. Finally, the authors would like to thank Rhonda Reinert of Technically Write for her careful review and extensive editing of this document and her many valuable suggestions for improving the organization and clarity of this document.

# Contents


**1 Introduction . . . . . 11**

1.1 Overall Input Structure . . . . . 12

1.2 Conventions for Command Descriptions . . . . . 15

1.2.1 Key Words . . . . . 15

1.2.2 User-Specified Input . . . . . 15

1.2.3 Optional Input . . . . . 16

1.2.4 Default Values . . . . . 16

1.2.5 Multiple Options for Values . . . . . 16

1.3 Style Guidelines . . . . . 18

1.3.1 Comments . . . . . 18

1.3.2 Continuation Lines . . . . . 18

1.3.3 Case . . . . . 18

1.3.4 Commas . . . . . 19

1.3.5 Blank Spaces . . . . . 19

1.3.6 General Format of the Command Lines . . . . . 19

1.3.7 Delimiters . . . . . 20

1.3.8 Order of Commands . . . . . 20

1.3.9 Abbreviated END Specifications . . . . . 20

1.3.10 Indentation . . . . . 21

1.4 Naming Conventions Associated with the Exodus II Database . . . . . 22

1.5 Major Scope Definitions for a Presto Input File . . . . . 23

**2 Command Descriptions . . . . . 24**

2.1 Utility/General Commands . . . . . 26

2.1.1 SIERRA Command Block . . . . . 26

2.1.2 Title . . . . . 26

2.1.3 Restart Control . . . . . 26

2.1.3.1 Restart Time . . . . . 27

2.1.3.2 Automatic Restart . . . . . 28

2.1.4 Functions . . . . . 29

2.1.5 Axes, Directions, and Points . . . . . 30

2.2 Materials . . . . . 32

2.2.1 Elastic Model . . . 33
2.2.2 Elastic-Plastic Model . . . 34
2.2.3 Elastic-Plastic Power-Law Hardening Model . . . 35
2.2.4 Soil and Crushable Foam Model . . . 36
2.2.5 Foam Plasticity Model . . . 38
2.2.6 Orthotropic Crush Model . . . 40
2.2.7 Orthotropic Rate Model . . . 42
2.2.8 Mie-Gruneisen Model . . . 44
2.2.9 Mie-Gruneisen Power-Series Model . . . 45
2.2.10 JWL (Jones-Wilkins-Lee) Model . . . 46
2.2.11 Ideal Gas Model . . . 47
2.3 Finite Element Model . . . 49
2.3.1 Definition of Material Model . . . 51
2.3.2 Element Strain Formulation . . . 52
2.3.3 Linear and Quadratic Bulk Viscosity . . . 52
2.3.4 Hourglass Control . . . 52
2.3.5 Membrane Scale Thickness . . . 52
2.3.6 Control Parameters for Shell Elements . . . 53
2.3.7 Truss Area . . . 54
2.3.8 Damper Area . . . 54
2.3.9 Energy Deposition . . . 54
2.3.10 Element Numerical Formulation . . . 55
2.3.11 Deactivate All Elements in an Element Block . . . 56
2.4 Presto Region and Procedure . . . 57
2.4.1 Presto Procedure . . . 57
2.4.2 Time Control . . . 57
2.4.2.1 Command Blocks for Time Control and Time Stepping . . . 57
2.4.2.2 Initial Time Step . . . 59
2.4.2.3 Time Step Scale Factor . . . 59
2.4.2.4 Time Step Increase Factor . . . 59
2.4.2.5 Step Interval . . . 60
2.4.2.6 Example . . . 61
2.4.3 Presto Region . . . 61

2.5 Use Finite Element Model. . . . . 62
2.6 Kinematic Boundary Conditions. . . . . 63
2.6.1 Fixed Displacement Components . . . . . 63
2.6.2 Prescribed Displacement. . . . . 63
2.6.3 Prescribed Velocity. . . . . 64
2.6.4 Prescribed Acceleration . . . . . 65
2.6.5 Fixed Rotation. . . . . 65
2.6.6 Prescribed Rotation. . . . . 66
2.7 Initial Conditions. . . . . 67
2.7.1 Initial Velocity Direction . . . . . 67
2.7.2 Initial Angular Velocity . . . . . 67
2.8 Force Boundary Conditions . . . . . 69
2.8.1 Pressure. . . . . 69
2.8.2 Prescribed Force . . . . . 69
2.8.3 Prescribed Moment. . . . . 70
2.9 Specialized Boundary Conditions. . . . . 71
2.9.1 Gravity . . . . . 71
2.9.2 Cavity Expansion . . . . . 71
2.9.3 Periodic. . . . . 73
2.9.4 Silent Boundary . . . . . 75
2.9.5 Spot-Weld. . . . . 75
2.10 Constraints. . . . . 78
2.10.1 Constant Distance Constraint . . . . . 78
2.10.2 Hex Shell Constraint. . . . . 78
2.11 Mass Property Calculations. . . . . 80
2.12 Contact . . . . . 81
2.12.1 Contact Definition Block . . . . . 82
2.12.2 Descriptions of Contact Surfaces . . . . . 83
2.12.2.1 Contact Surface. . . . . 83
2.12.2.2 Contact All Blocks . . . . . 84
2.12.3 Remove Initial Overlap. . . . . 84
2.12.4 Angle for Multiple Interactions . . . . . 85
2.12.5 Iterative Enforcement . . . . . 85

2.12.6 Friction Models . . . 85
2.12.6.1 Frictionless Model . . . 86
2.12.6.2 Constant Friction Model . . . 86
2.12.6.3 Tied Model . . . 86
2.12.7 Analytic Contact Surfaces . . . 86
2.12.7.1 Plane . . . 86
2.12.7.2 Cylinder . . . 87
2.12.7.3 Sphere . . . 88
2.12.8 Default Values for Interactions . . . 88
2.12.8.1 Normal and Tangential Tolerance . . . 89
2.12.8.2 Normal and Tangential Overlap Tolerance . . . 90
2.12.8.3 Friction Model . . . 90
2.12.8.4 Automatic Kinematic Partition . . . 91
2.12.9 Values for Specific Interactions . . . 91
2.12.9.1 Surface Identification . . . 92
2.12.9.2 Kinematic Partition . . . 93
2.12.10 Example . . . 93
2.13 Results Output . . . 95
2.13.1 Output Nodal Variables . . . 96
2.13.2 Output Element Variables . . . 97
2.13.3 Output Global Variables . . . 97
2.13.4 Set Begin Time for Results Output . . . 98
2.13.5 Adjust Interval for Time Steps . . . 98
2.13.6 Output Interval Specified by Time Increment . . . 98
2.13.7 Additional Times for Output . . . 98
2.13.8 Output Interval Specified by Step Increment . . . 98
2.13.9 Additional Steps for Output . . . 99
2.13.10 Set End Time for Results Output . . . 99
2.14 History Output . . . 100
2.14.1 Output Variables . . . 101
2.14.1.1 Nodal and Element Output Variables . . . 101
2.14.1.2 Global Output Variables . . . 101
2.14.2 Set Begin Time for History Output . . . 102
2.14.3 Adjust Interval for Time Steps . . . 102

2.14.4 Output Interval Specified by Time Increment . . . . . 102
2.14.5 Additional Times for Output. . . . . 102
2.14.6 Output Interval Specified by Step Increment . . . . . 103
2.14.7 Additional Steps for Output . . . . . 103
2.14.8 Set End Time for History Output . . . . . 103
2.15 Restart Data . . . . . 104
2.15.1 Set Begin Time for Restart Writes . . . . . 105
2.15.2 Adjust Interval for Time Steps . . . . . 105
2.15.3 Restart Interval Specified by Time Increment . . . . . 106
2.15.4 Additional Times for Restart . . . . . 106
2.15.5 Restart Interval Specified by Step Increment . . . . . 106
2.15.6 Additional Steps for Restart . . . . . 106
2.15.7 Set End Time for Restart Writes. . . . . 106
**3 Example Problem. . . . . 107**
**References . . . . . 114**
**Appendix A: Command Specification . . . . . 116**
**Appendix B: Registered Variables. . . . . 127**
**Index . . . . . 132**

# Figures

2.1 Periodic structure. . . . . 73
2.2 Force-displacement curve for spot-weld. . . . . 76
2.3 Illustration of normal and tangential tolerances. . . . . 89
3.1 Mesh for example problem: (a) Box (blue and green surfaces) with plate in top (red surface) and (b) Mesh with blue and red surfaces removed to show internal spheres (yellow). . . . . 107
3.2 Results of Crush 124 Spheres test. . . . . 107

# Presto User's Guide Version 1.05

## 1 Introduction

Presto is a three-dimensional transient dynamics code with a versatile element library, nonlinear material models, large deformation capabilities, and contact. It is built on the SIERRA Framework [1]. SIERRA provides a data management framework in a parallel computing environment that allows the addition of capabilities in a modular fashion. Contact capabilities are parallel and scalable; these capabilities are provided by ACME [2].

This document describes how to create an input file for Presto. Highlights of the document contents are as follows:

- Section 1 describes the overall structure of the input file, including conventions for the command descriptions, style guidelines for file preparation, and naming conventions for input files that reference the Exodus II database [3]. The section concludes with an example of the general structure of an input file employing the concept of scope.
- Section 2 provides detailed descriptions of the commands that can be used in a Presto input file. These command descriptions are grouped in sections based on their functionality. For example, the commands for kinematic boundary conditions are described in one section; the commands for contact definition are described in another section.
- Section 3 provides a sample input file from an analysis of 16 lead spheres being crushed together inside a steel box.
- References lists the sources cited in this document.
- Appendix A gives all of the permissible Presto input lines in their proper scope.
- Appendix B contains the registered variables that can be selected for output in the results file.

## 1.1 Overall Input Structure

Presto is only one of the codes built on the SIERRA Framework. The SIERRA Framework provides the capability to perform multiphysics analyses using a number of codes built on the SIERRA Framework. Input files may be constructed for analyses using only Presto, or input files may be constructed for analyses using Presto and some other analysis code built on the SIERRA Framework. For example, you might run Adagio [4], the quasti-static structural response code, to compute a stress state, and then pass the results of this analysis to Presto as initial conditions for the Presto analysis. For a multiphysics analysis using Presto and Adagio, the input commands for the analysis will appear in a single input file. The time-step control, the mesh-related definitions, and the boundary conditions for both Presto and Adagio will all be in the same input file. Therefore, the input for Presto reflects the fact that a Presto analysis could be part of a multiphysics analysis.

To create files defining multiphysics analyses, the input files use a concept called scope. Scope is used to group similar commands; a scope can be nested inside another scope. The broadest scope in the input file is the domain scope. The domain scope contains information that is physics independent. Examples of physics-independent information are definitions of functions and materials. Thus, in our above example of a Presto/Adagio multiphysics analysis, both Adagio and Presto could reference functions to define such things as time histories for boundary conditions or stress-strain curves. Some of the functions could even be shared by these two applications. Both Presto and Adagio would share information about materials. These codes would reference the same definitions of material models.

Within the domain scope are two other important scopes—the procedure scope and the region scope. For a multiphysics analysis, the domain scope could contain several different procedures and several different regions. For Presto, the procedure scope controls the overall analysis from the start time to the end time. The region scope controls a single time step. The region is nested inside the procedure, and the procedure is nested inside the domain. In addition to the region scope, the procedure scope also contains a scope for controlling the time steps. This time-step control scope sets the start and end times for the analysis, and is nested inside the procedure scope but outside the region scope.

Inside the region scope for Presto are such things as definitions for boundary conditions and contact. The mesh is also specified in the region. In a multiphysics analysis, there would be more than one region. In our Presto/Adagio example, there would be both a Presto region and an Adagio region. The definitions for boundary conditions and contact and the mesh specification for Presto would appear in the Presto region; the definitions for boundary conditions and contact and the mesh specification for Adagio would appear in the Adagio region.

The input for Presto consists of command blocks and command lines. The command blocks define a scope. These command blocks group command lines or other command blocks that share a similar functionality. A command block will begin with an input line that has the word “begin”; the command block will end with an input line that has the

word “end”. The domain scope, for example, is defined by a command block that begins with an input line of the form

```
BEGIN SIERRA my_problem .
```

The two character strings `BEGIN` and `SIERRA` are the key words for this command block. An input line defining a command block or command line will have one or more key words. The string `my_problem` is a user-specified name for this domain scope. The domain scope is terminated by an input line of the form

```
END SIERRA my_problem ,
```

where `END` and `SIERRA` are the key words to end this command block. If all of the scopes are properly nested, the domain scope can also be terminated simply by using

```
END .
```

This abbreviated command line will be discussed in more detail in later sections. There are similar input lines used to define the procedure and region scopes. Boundary conditions are another example where a scope is defined. A particular instance of a boundary condition for a prescribed displacement boundary condition is defined with a command block. The command block for the boundary condition begins with an input line of the form

```
BEGIN PRESCRIBED DISPLACEMENT
```

and ends with an input line of the form

```
END PRESCRIBED DISPLACEMENT
```

or just simply

```
END .
```

Command lines appear within the command blocks. The command lines typically have the form `keyword = value`, where `value` can be a real, an integer, or a string. In the previous example of the prescribed displacement boundary condition, there would be command lines inside the command block that are used to set various values. For example, the boundary condition might apply to all nodes in node set 10, in which case there would be a command line of the form

```
NODE SET = nodelist_10 .
```

If the prescribed displacement were to be applied along a given component direction, there would be a command line of the form

```
COMPONENT = X ,
```

which would specify that the prescribed displacement would be in the *x*-direction. Finally, if the displacement magnitude is described by a time history function with the name `cosine_curve`, there would a command line of the form

```
FUNCTION = cosine_curve .
```

The command block for the boundary condition with the appropriate command lines would appear as follows:

```
BEGIN PRESCRIBED DISPLACEMENT
  NODE SET = nodelist_10
  COMPONENT = X
  FUNCTION = cosine_curve
END PRESCRIBED DISPLACEMENT .
```

It is possible to have a command line with the same key words appearing in different scopes. For example, we might have a command line identified by the word `TYPE` in two or more different scopes. The command line would perform different functions based on the scope in which it appeared, and the associated value could be different in the two locations.

The input lines are read by a parser that searches for recognizable key words. If the key words in an input line are not in the list of key words used by Presto to describe command blocks and command lines, the parser will generate an error. A set of key words defining a command line or command block for Presto that is not in the correct scope will also cause the parser to generate an error. For example, the key words `STEP INTERVAL` define a valid command line in the scope of the `TIME CONTROL` command block. However, if this command line were to appear in the scope of one of the boundary conditions, it would not be in the proper scope and the parser would generate an error. Once the parser has an input line with any recognizable key words in the proper scope, a method can be called that will handle the input line.

## 1.2 Conventions for Command Descriptions

The conventions below are used to describe the input commands for Presto. NOTE: In this document, all of the sentences containing input lines are punctuated correctly. For example, the function command line in a sentence such as this one would appear as

```
FUNCTION = <string>function_name .
```

The space after `function_name` indicates that the period following this space is not a part of the command line but is the correct punctuation in the text. The above command line in the input file would NOT have a period.

A number of the individual command lines discussed in the text appear on several text lines. In the text of this user's guide, the continuation symbols that are used to continue lines in an actual input file (`/#` and `/$`, Section 1.3.2) are not used for those instances where the description of the command line appears on several text lines. The description of command lines will clearly indicate all of the key words, delimiters, and values that constitute a complete command line. As an example, the `DEFINE POINT` command line (Section 2.1.5) is presented in the text as

```
DEFINE POINT <string>point_name WITH COORDINATES
    <real>value_1 <real>value_2 <real>value_3 .
```

If the `DEFINE POINT` command line were used as a command line in an input file and spread over two input lines, it would appear, with actual values, as

```
DEFINE POINT center WITH COORDINATES /#
10.0 144.0 296.0 ,
```

where the `/#` symbol implies the first line is continued onto the second line.

### 1.2.1 Key Words

The key word or key words for a command are shown in uppercase letters. For actual input, you can use all uppercase letters for the key words, all lowercase letters for the key words, or some combination of uppercase and lowercase letters for the key words.

### 1.2.2 User-Specified Input

The input that you supply is shown in lowercase letters. The user-supplied input may be a real number, an integer, or a string. For the command descriptions, a type appears before the user input. The type (real, integer, string) description is enclosed by angle brackets, <>, and precedes the user-supplied input. For example,

```
<real>value
```

indicates that the quantity `value` is a real. For the description of an input command, you would see

```
FUNCTION = <string>function_name .
```

Your input would be

```
FUNCTION = my_name
```

if you have specified a function name called `my_name`.

Valid user input consists of the following:

| | |
|---|---|
| `<integer>` | Integer data is a single integer number. |
| `<real>` | Real data is a single real number. It may be formatted with the usual conventions, such as `1234.56` or `1.23456e+03`. |
| `<string>` | String data is a single string. |
| `<string list>` | A string list consists of multiple strings separated by white space, a comma, or white space combined with a comma. |

### 1.2.3 Optional Input

Anything enclosed by square brackets, [ ], in the input line descriptions represents optional input.

### 1.2.4 Default Values

A value enclosed by parentheses, (), appearing after the user input denotes the default value. For example,

```
SCALE FACTOR = <real>scale_factor(1.0)
```

implies the default value for `scale_factor` is 1.0. Any value you specify will overwrite the default.

### 1.2.5 Multiple Options for Values

Quantities separated by the | symbol indicate that one and only one of the possible choices must be selected. For example,

```
EXPANSION RADIUS = <string>SPHERICAL|CYLINDRICAL
```

implies that expansion radius must be defined as `SPHERICAL` or `CYLINDRICAL`. At least one value must appear. This convention also applies to some of the command options within a begin/end block. For example,

```
SURFACE = <string>surface_name |
  NODE SET = <string>nodelist_name
```

in a command block specifies that either a surface or node set must be specified.

Quantities separated by the / symbol can appear in any combination, but any one quantity in the sequence can appear only once. For example,

```
COMPONENTS = <string>X/Y/Z
```

implies that components can equal any combination of X, Y, and Z. Any value (X or Y or Z) can appear at most once, and at least one value of X, Y, or Z must appear. Some examples of valid expressions in this case are

```
COMPONENTS = Z ,
COMPONENTS = Z X ,
COMPONENTS = Y X Z ,
```

and

```
COMPONENTS = Z Y X .
```

An example of an invalid expression would be

```
COMPONENTS = Y Y Z .
```

## 1.3 Style Guidelines

This section gives information that will affect the overall organization and appearance of your input file. It also contains recommendations that will help you construct input files that are readable and easy to proof.

### 1.3.1 Comments

A comment is anything between the # symbol or the $ symbol and the end-of-line. If the first nonblank character in a line is a # or $, the entire line is a comment line. You can also place a # or $ (preceded by a blank space) after the last character in an input line used to define a command block or command line.

### 1.3.2 Continuation Lines

An input line can be continued by placing a \# pair of characters or by the \$ at the end of the line. The following line is then taken to be a continuation of the preceding line that was terminated by the \# or \$. Note that everything after the line-continuation pair of characters is discarded, including the end-of-line.

### 1.3.3 Case

Almost all of the character strings in the input lines are case insensitive. For example, the `BEGIN SIERRA` key words could appear as

```
BEGIN SIERRA
```

or

```
begin sierra
```

or

```
Begin Sierra .
```

You could specify a `SIERRA` command block with

```
BEGIN SIERRA BEAM
```

and terminate the command block with

```
END SIERRA beam .
```

There are a few exceptions where case is important. For example, in the specification of variable names for results output there are some variable names with a mixture of lowercase and uppercase letters. Check the appendices on registered variables. Also, specifications of file names are case sensitive. If you have defined a restart file with uppercase and lowercase letters and want to use this file for a restart, the file name you use to request this restart file must exactly match the original definition you chose.

### 1.3.4 Commas

Commas in input lines are ignored.

### 1.3.5 Blank Spaces

We highly recommend that everything be separated by blank spaces. For example, a command line of the form

```
node set = nodelist_10
```

is recommended over

```
node set= nodelist_10
```

or

```
node set =nodelist_10 .
```

Both of the above two lines are correct, but it is easier to check the first form (the equal sign surrounded by blank space) in a large input file.

The parser will accept the line

```
BEGIN SIERRABEAM ,
```

but it is harder to check this line for the correct spelling of the key words and the intended domain name than the line

```
BEGIN SIERRA BEAM .
```

It is possible to introduce hard-to-detect errors because of the way in which the blank spaces are handled by the command parser. Suppose you type

```
begin definition for functions my_func
```

rather than the correct form, which is

```
begin definition for function my_func .
```

For the incorrect form of this command line (in which `functions` is used rather than `function`), the parser will generate a string name of

```
s my_func
```

for the function name rather than the expected name of

```
my_func.
```

If you attempt to use a function named `my_func`, the parser will generate an error because the list of function names will include `s my_func` but not `my_func`.

### 1.3.6 General Format of the Command Lines

In general, command lines have the form

```
keyword = value .
```

This pattern is not always followed, but it describes the vast majority of the command lines.

### 1.3.7 Delimiters

We recommend that you use only the = sign when a delimiter is required. For most command lines, you can actually use =, `is`, or `are` interchangeably as a delimiter. For command lines with a delimiter, you could specify

```
components = X ,
```

or

```
components is X ,
```

or

```
components are X .
```

The = sign is strongly recommended as the delimiter of choice. It provides a strong visual cue for separating key words from values. By relying on the = sign as a delimiter, it will be much easier to proof your input file. It will also make it easier to do “cut and paste” operations. If you accidently delete an = sign, it is much easier to detect than accidently removing part of an `is` or `are` delimiter.

### 1.3.8 Order of Commands

There are no requirements for ordering the commands. Both the input sequence

```
BEGIN PRESCRIBED DISPLACEMENT
   NODE SET = nodelist_10
   COMPONENT = X
   FUNCTION = cosine_curve
END PRESCRIBED DISPLACEMENT
```

and the input sequence

```
BEGIN PRESCRIBED DISPLACEMENT
   FUNCTION = cosine_curve
   COMPONENT = X
   NODE SET = nodelist_10
 END PRESCRIBED DISPLACEMENT
```

are valid, and they produce the same result. REMEMBER, however, that command lines and command blocks must appear in the proper scope.

### 1.3.9 Abbreviated END Specifications

It is possible to terminate a command block without including the key word or key words that identify the block. For example, you could define a specific instance of the prescribed displacement boundary condition with

```
BEGIN PRESCRIBED DISPLACEMENT
```

and terminate it simply with

```
END
```

as opposed to

```
END PRESCRIBED DISPLACEMENT .
```

Both the short termination (`END` only) and the long termination (`END` followed by identification, or name, of the command block) are valid. It is recommended that the long termination be used for any command block that becomes large. For example, the `RESULTS OUTPUT` command block described in later sections can become fairly lengthy, so this is probably a good place to use the long termination. For most boundary conditions, the command block will typically consist of five lines. In such cases, the short termination can be used. Using the long termination for the larger command blocks will make it easier to proof your input files.

### 1.3.10 Indentation

When constructing an input file, it is useful to indent a scope that is nested inside another scope. Command lines within a command block should also be indented in relation to the lines defining the command block. This will make it easier to construct the input file with everything in the correct scope and with all of the command blocks in the correct structure.

## 1.4 Naming Conventions Associated with the Exodus II Database

When the mesh file has an Exodus II format, there are three basic conventions that apply to user input for various command lines. First, for a mesh file with the Exodus II format, the Exodus II side set is referenced as a surface. In SIERRA, a surface consists of faces on the surface of an element block plus all of the nodes and edges associated with these faces. A surface definition can be used not only to select a group of faces but also to select a group of edges or a group of nodes that are associated with those faces. In the case of boundary conditions, a surface definition can be used not only to apply boundary conditions that typically use surface specifications (pressure) but also to apply boundary conditions for what are referred to as nodal boundary conditions (fixed displacement components). For nodal boundary conditions that use the surface specification, all of the nodes associated with the faces on a specific surface will have this boundary condition applied to them. The specification for a surface identifier in the following sections is `surface_name`. It typically has the form `surface_integerid`, where `integerid` is the integer identifier for the surface. If the side set is 125, the value of `surface_name` would be `surface_125`. It is also possible to generate an alias for the side set and use this for `surface_name`. If `surface_125` is aliased to `outer_skin`, then `surface_name` becomes `outer_skin` in the actual input line.

Second, for a mesh file with the Exodus II format, the Exodus II node set is still referenced as a node set. A node set can be used only for cases where a group of nodes needs to be defined. The specification for a node set identifier in the following sections is `nodelist_name`. It typically has the form `nodelist_integerid`, where `integerid` is the integer identifier for the node set. If the node set is 225, the value of `nodelist_name` would be `nodelist_225`. It is also possible to generate an alias for the node set and use this for `nodelist_name`. If `nodelist_225` is aliased to `inner_skin`, then `nodelist_name` becomes `inner_skin` in the actual input line.

Third, an element block remains a block. The specification for an element block identifier in the following sections is `block_name`. It typically has the form `block_integerid`, where `integerid` is the integer identifier for the block. If the element block is 300, the value of `block_name` would be `block_300`. It is also possible to generate an alias for the block and use this for `block_name`. If `block_300` is aliased to `big_chunk`, then `block_name` becomes `big_chunk` in the actual input line.

## 1.5 Major Scope Definitions for a Presto Input File

The typical Presto input file will have the structure shown below. The major scopes—domain, procedure, and region—are delineated with input lines for command blocks. Comment lines are included that indicate some of the key scopes that will appear within the major scopes. Note the indentation used for this example.

```
BEGIN SIERRA <string>some_name
  #
  # All command blocks and command lines in the domain
  # scope appear here. The PROCEDURE PRESTO command
  # block is the beginning of the next scope.
  #
  # function definitions
  # material descriptions
  # description of mesh file
  #
  BEGIN PROCEDURE PRESTO <string>procedure_name
    #
    # time step control
    #
    BEGIN REGION PRESTO <string>region_name
      #
      # All command blocks and command lines in the
      # region scope appear here.
      #
      # specification for output of result
      # specification for restart
      # boundary conditions
      # definition of contact
      #
    END [REGION PRESTO <string>region_name]
  END [PROCEDURE PRESTO <string>procedure_name]
END [SIERRA <string>some_name]
```

# 2 Command Descriptions

This section describes the commands recognized by the parser in Presto. The commands are grouped by functionality, as follows:

**2.1 Utility/General Commands**. These general command blocks and command lines appear only in the domain scope. They will define such things as material models, functions, and special geometric entities.

**2.2 Materials**. These command blocks and command lines appear only in the domain scope. `PROPERTY SPECIFICATION FOR MATERIAL` command blocks define all of the parameters for one or more material models.

**2.3 Finite Element Model**. These command blocks and command lines appear only in the domain scope. A `FINITE ELEMENT MODEL` command block provides the description of a mesh that will be associated with the analysis. Embedded in this command block are descriptions for each of the element blocks.

**2.4 Presto Region and Procedure**. These command blocks specify the region and the procedure. The region scope is nested inside the procedure scope, and the procedure scope is nested inside the domain scope. The `TIME CONTROL` command block is nested inside the procedure scope (but outside the region scope).

**2.5 Use Finite Element Model**. This command specifies the model (described in Section 2.3) to be used in the region.

**2.6 Kinematic Boundary Conditions**. These command blocks and command lines appear only in the region scope. They define a variety of kinematic boundary conditions.

**2.7 Initial Conditions**. These command blocks and command lines appear only in the region scope. They define several initial conditions.

**2.8 Force Boundary Conditions**. These command blocks and command lines appear only in the region scope. They define a variety of force boundary conditions.

**2.9 Specialized Boundary Conditions**. These command blocks and command lines appear only in the region scope. They define a number of specialized boundary conditions such as those for nonreflecting surfaces, periodic structures, and cavity expansion.

**2.10 Constraints**. These command blocks and command lines appear only in the region scope. They define constraint relations.

**2.11 Mass Property Calculations**. A command block for this functionality appears only in the region scope. The command lines associated with this block specify mass

property calculations for a single element block or a structure defined by a group of element blocks.

**2.12 Contact**. These command blocks and command lines appear only in the region scope. They define various input parameters for controlling the contact algorithm (ACME).

**2.13 Results Output**. These command blocks and command lines appear only in the region scope. They control the type of information written to results files and the frequency at which this information is written.

**2.14 History Output**. These command blocks and command lines appear only in the region scope. They control the type of information written to history files and the frequency at which this information is written.

**2.15 Restart Data**. These command blocks and command lines appear only in the region scope. They control the frequency at which restarts are written.

A complete list of commands within their proper scope is given in Appendix A.

## 2.1 Utility/General Commands

These commands are used to set up some of the fundamentals of the Presto input. The commands are physics independent, or at least can be shared between physics. The commands lie in the domain scope, not in the procedure or region scope.

### 2.1.1 SIERRA Command Block

```
BEGIN SIERRA <string>name
  #
  # All other command blocks and command lines
  # appear within the domain scope defined by
  # begin/end sierra.
  #
END [SIERRA <string>name]
```

All input commands for Presto must occur within a `SIERRA` command block. The syntax for beginning the command block is

```
BEGIN SIERRA <string>name
```

and for terminating the command block is

```
END [SIERRA <string>name] ,
```

where `name` is a name for the `SIERRA` command block. All other commands for the analysis must be within this command block structure. The name for the `SIERRA` command block is often a descriptive name that identifies the analysis. The name is not currently used anywhere else in the file and is completely arbitrary.

### 2.1.2 Title

```
TITLE <string list>title
```

To permit a fuller description of the analysis, the Presto input has a `TITLE` command line for the analysis, where `title` is a text description of the analysis.

This title is not currently used in the analysis. It simply appears in the input file for added information about the analysis.

### 2.1.3 Restart Control

The restart capability in Presto allows a user to run an analysis up to a certain time, stop the analysis at that time, and then restart the analysis at this stop time. The use of the restart capability involves commands in **both the domain scope and the region scope**. One of two restart command lines (`RESTART` or `RESTART TIME`) in the domain scope is used to select a time at which a restart is to begin. A command block in the region scope (`RESTART DATA`) specifies restart file names and the frequency at which the restart files will be written. The `RESTART DATA` command block is described in Section 2.15. This section discusses the command lines that appear in the domain scope.

When using the restart capability, you can reset a number of the parameters in the input file. Not all parameters, however, can be reset. Users should exercise care in resetting parameters in the input file for a restart.

There are two methods to initiate a restart with Presto. First, you can specify a time from a specific restart file for the restart. This method uses the `RESTART TIME` command line described in Section 2.1.3.1. Second, you can specify an automatic restart feature. This method uses the `RESTART` command line described in Section 2.1.3.2. The command lines for both of these methods are in the **domain scope**. All other commands for restart are in the **region scope** in the `RESTART DATA` command block.

For restarts specified with a restart time from a specific restart file, the user will have to be concerned about overwriting information in existing files. The issue of overwriting information is discussed in Section 2.1.3.1.

Restart can be used to break a long-running analysis into several smaller runs so that the user can examine intermediate results before proceeding with the next step. The examples in the Section 2.1.3.1 describe this use of restart. Restart can also be used in case of abnormal termination. If a restart file has been written at various intervals throughout the analysis up to the point where the termination has occurred, you can pick some restart time before the termination and restart the problem from there. Thus, users do not have to go back to the beginning of the analysis, but can continue the analysis at some time well into the analysis.

The amount of data written at a restart time is quite large. The restart data written at a given time is a complete description of the state for the problem at that time. The restart data includes not only information such as displacement, velocity, and acceleration, but also information such as elements stresses and and all of the state variables for the material model associated with each element.

### 2.1.3.1 Restart Time

```
RESTART TIME = <real>restart_time
```

The `RESTART TIME` command line is used to specify a time from a specific restart file for the restart run.

To understand how the `RESTART TIME` command line is used, assume that an initial run (with no restart time specified) is made with termination time $T_0$. For this initial run, a restart file is written at time $T_0$ (and possibly at other times). The restart file can be used in a second run. In the second run with restart, the user specifies a new termination time $T_1$, with $T_1 > T_0$. The input file for the restart run will include the `RESTART TIME` command line with a time less than or equal to $T_0$. This process may be repeated any number of times to do a sequence of restarts.

This restart option will pick the restart time on the restart file that is closest to the user-specified time on the `RESTART TIME` command line. If the user specifies a restart time greater than the last time written to a restart file, then the last time written to the restart file is picked as the restart time.

For this first run, assume that the restart file is called `beam.r`. If the name of the restart file has not been changed in the input for the run from $T_0$ to $T_1$, the second run will automatically generate a new restart file `beam.r-s0002`. Now suppose that a new restart is initiated to run to time $T_2$, with $T_2 > T_1$. If, as before, the name of the restart file has not been changed in the input for the run from $T_1$ to $T_2$, the third run will use the same restart file named `beam.r-s0002`. The restart information from time $T_1$ to $T_2$ will overwrite the restart information from time $T_0$ to $T_1$. By using the automatic restart feature described in the next section, however, you can force the restart files to be incremented automatically so that the restart file from $T_1$ to $T_2$ would be `beam.r-s0003`. A similar pattern for file suffixes also occurs for the results output files and the history files if the names for these files are not changed in the input file for the various runs up to time $T_2$.

To prevent the overwriting of existing restart files, you can specify both an input restart file and an output restart file and rename the results output and history files for the various restarts. The use of the `RESTART TIME` command line requires the user to be more active in the management of the file names to prevent the overwriting of information. The automatic restart feature prevents the overwrite of restart, results output, and history files.

#### 2.1.3.2 Automatic Restart

```
RESTART = AUTOMATIC
```

The `RESTART` command line automatically selects for restart the last restart time written to the last restart file.

To understand how the `RESTART` command line is used, assume an initial run (with no restart time specified) is made with termination time $T_0$. For this first run, a restart file is written with at least one requested restart time equal to $T_0$. For this first run, assume that the restart file has been called `beam.r`. In the second run with restart, the user will specify a new termination time $T_1$, with $T_1 > T_0$. The restart run will include the `RESTART` command line. The second run will start at time $T_0$ even if there are restart times less than $T_0$ written to the restart file. If the name of the restart file has not been changed in the input for the run from $T_0$ to $T_1$, the second run will automatically generate a new restart file named `beam.r-s0002`. Suppose the last restart written to `beam.r-s0002` is at time $T_1$. If a new restart is initiated to run to time $T_2$, with $T_2 > T_1$, and the automatic restart option is used, the new restart begins at time $T_1$. If, as before, the name of the restart file

has not been changed in the input for the run from $T_1$ to $T_2$, the third run will automatically generate a new restart file named `beam.r-s0003`. This process may be repeated any number of times.

The automatic increment feature applied to the restart file names is also applied to the results output files and the history files. If our first results output file is `beam.e` and the name of the results output file is not changed in the input files for the restarts, the subsequent results output files will be `beam.e-s0002`, `beam.e-s0003`, etc. A similar pattern will hold for history files. If our first history file is `beam.h`, the subsequent history files will be `beam.h-s0002`, `beam.h-s0003`, etc. if the name of the history file is not changed in the input files for the restarts.

The automatic restart feature lets the user restart runs with minimal changes to the input file. The only quantity that must be changed to move from one restart to another is the termination time.

### 2.1.4 Functions

```
BEGIN DEFINITION FOR FUNCTION <string>function_name
  TYPE = <string>CONSTANT | PIECEWISE LINEAR
  [ABSCISSA = <string>abscissa_label]
  [ORDINATE = <string>ordinate_label]
  BEGIN VALUES
    <real>value_1   [<real>value_2
    <real>value_3    <real>value_4
    ...              <real>value_n]
  END [VALUES]
END [DEFINITION FOR FUNCTION <string>function_name]
```

A number of Presto features are driven by a user-defined description of the dependence of one variable on another. For instance, the prescribed displacement boundary condition requires the definition of a time-versus-displacement relation, and the thermal strain computations require the definition of a thermal-strain-versus-temperature relation. SIERRA provides a general method of defining these relations as functions using the `DEFINITION FOR FUNCTION` command block, as shown above.

The values in the above input lines of this command block are as follows:

- The string `function_name` is a name for the function that is unique to the function definitions within the input file. This name is used to refer to this function in other locations in the input file.
- The string `function_type` defines the type of the function. The value of `function_type` can be `CONSTANT` or `PIECEWISE LINEAR`.
- The string `abscissa_label` is an optional label that describes the independent variable (*x*-axis).
- The string `ordinate_label` is an optional label that describes the dependent variable (*y*-axis).

- The real values `value_1` through `value_n` describe the function. For a constant function, only one value is needed. For a piecewise linear function, the values are (*x*, *y*) pairs of data that describe the function. The values are nested inside a `VALUES` command block.

For any value greater than the last abscissa value in the function, the last ordinate value is used for the function value. For example, suppose we have a piecewise linear function that indicates a function `my_func` describes a time history for a pressure load where the pressure increases from 0 to 50,000 psi from time 0.0 sec to time $1.0 \times 10^{-3}$ sec. The last time specified in the function is $1.0 \times 10^{-3}$. Now, suppose our final analysis time is $2.0 \times 10^{-3}$ sec. Then, from the time $1.0 \times 10^{-3}$ to the time $2.0 \times 10^{-3}$, the value for this function (`my_func`) will be 50,000 psi.

There is no limit to the number of functions that can be defined. All function definitions must appear within the domain scope.

### 2.1.5 Axes, Directions, and Points

```
DEFINE POINT <string>point_name WITH COORDINATES
    <real>value_1 <real>value_2 <real>value_3
DEFINE DIRECTION <string>direction_name WITH VECTOR
    <real>value_1 <real>value_2 <real>value_3
DEFINE AXIS <string>axis_name WITH POINT
    <string>point_1 POINT <string>point_2
DEFINE AXIS <string>axis_name WITH POINT
    <string>point DIRECTION <string>direction
```

A number of Presto features require the definition of geometric entities. For instance, the prescribed displacement boundary condition requires a direction definition, and the cylindrical velocity initial condition requires an axis definition. Currently, Presto input permits the definition of points, directions, and axes. Definition of these geometric entities occurs in the domain scope.

The `DEFINE POINT` command line is used to define a point:

```
DEFINE POINT <string>point_name WITH COORDINATES
    <real>value_1 <real>value_2 <real>value_3  ,
```

*where*

- The string `point_name` is a name for this point. This name must be unique to all other points defined in the input file.
- The real values `value_1`, `value_2`, and `value_3` are the *x*, *y*, and *z* coordinates of the point.

The `DEFINE DIRECTION` command line is used to define a direction:

```
DEFINE DIRECTION <string>direction_name WITH VECTOR
    <real>value_1 <real>value_2 <real>value_3  ,
```

*where*

- The string `direction_name` is a name for this direction. This name must be unique to all other directions defined in the input file.
- The real values `value_1`, `value_2`, and `value_3` are the *x*, *y*, and *z* magnitudes of the direction vector.

There are two command lines that can be used to define an axis. The first DEFINE AXIS command line uses two points:

```
DEFINE AXIS <string>axis_name WITH POINT
    <string>point_1 POINT <string>point_2  ,
```

*where*

- The string `axis_name` is a name for this axis. This name must be unique to all other axes defined in the input file.
- The strings `point_1` and `point_2` are the names for two `points` defined in the input file.

The second DEFINE AXIS command line uses a point and a direction:

```
DEFINE AXIS <string>axis_name WITH POINT
    <string>point DIRECTION <string>direction  .
```

*where*

- The string `axis_name` is a name for this axis. This name must be unique to all other axes defined in the input file.
- The string `point` is the name of a point defined in the input file.
- The string `direction` is the name of a direction defined in the input file.

## 2.2 Materials

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  #
  # Command blocks and command lines for material
  # models appear in this scope.
  #
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

`PROPERTY SPECIFICATION FOR MATERIAL` command blocks appear in the domain scope in the general form shown above. These command blocks are physics independent in the sense that the information in them can be shared by more than one application. For example, the `PROPERTY SPECIFICATION FOR MATERIAL` command blocks contain density information that can be shared among several applications.

The command block begins with the input line

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
```

and is terminated with the input line

```
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name] ,
```

where the string `mat_name` is a user-specified name for the command block.

Within a `PROPERTY SPECIFICATION FOR MATERIAL` command block, there will be other command blocks and command lines that describe particular material models. These material models are described by a set of material-model command blocks that follow the naming convention of `PARAMETERS FOR MODEL model_name`, where `model_name` identifies the particular material model. Each such command block contains all of the parameters needed to describe a particular material model. NOTE: More than one material-model command block can appear within a `PROPERTY SPECIFICATION FOR MATERIAL` command block. Suppose we have a `PROPERTY SPECIFICATION FOR MATERIAL` command block called `steel`. It would be possible to have two material-model command blocks within this command block. One of the material-model command blocks would provide an elastic model for `steel`; the other material-mode command block would provide an elastic-plastic model for `steel`.

Both `PROPERTY SPECIFICATION FOR MATERIAL` command blocks and material-model command blocks are referenced by the element-block command block (also known as the `FINITE ELEMENT MODEL` command block), which is described in Section 2.3.

For information about the elastic, elastic-plastic, and elastic-plastic power-law hardening material models, consult Reference 5. For information about the orthotropic crush model, consult Reference 6. For information about the energy-dependent models (Mie-Gruneisen, Mie-Gruneisen power series, JWL, ideal gas), consult Reference 7.

For information about the soil and crushable foam model, consult with the Pronto3d document listed as Reference 8. The soil and crushable foam model in Presto is the same as the soil and crushable foam model in Pronto3d. The Pronto3d model is based on a

material model developed by Krieg [9]. The Krieg version of the soil and crushable foam model was later modified by Swenson and Taylor [10]. The soil and crushable foam model developed by Swenson and Taylor is the model in both Pronto3d and Presto.

For information about the foam plasticity model and orthotropic rate model, contact William Scherzinger at Sandia National Laboratories in Albuquerque, NM. His phone number is 505-284-4866, and his email address is wmscher@sandia.gov.

The material models that are most useful in Presto are described in Section 2.2.1 through Section 2.2.11. These models do not constitute the entire set of material models that are implemented in the SIERRA Framework. There are material models implemented in the SIERRA Framework that are useful for other solid mechanics codes, but not for Presto.

You should also be aware that not all of the material models available are applicable to all of the element types. As one example, there is a one-dimensional elastic material model that is used for a truss element but is not applicable to solid elements such as hexahedra or tetrahedra. For this particular example, the specific material-model usage is hidden from the user. If the user specifies a linear-elastic material model for a truss, the one-dimensional elastic material model is used. If the user specifies a linear-elastic material model for a hexahedron, a full three-dimensional elastic material model is used. As another example, the energy-dependent material models cannot be used for a one-dimensional element such as a truss. The energy-dependent material models can only be used for solid elements such as hexahedra and tetrahedra.

For each material model, the parameters needed to describe that model are listed in the section pertinent to that particular model. Solid models with elastic constants require only two elastic constants. These two constants are then used to generate all of the elastic constants for the model. For example, if the user specifies Young's modulus and Poisson's ratio, then the shear modulus, bulk modulus, and lambda are calculated. If the shear modulus and lambda are specified, then Young's modulus, Poisson's ratio, and the bulk modulus are calculated.

### 2.2.1 Elastic Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ELASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
  END [PARAMETERS FOR MODEL ELASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

An elastic material model is used to describe simple linear-elastic behavior of materials. This model is generally valid for small deformations.

For an elastic material, an elastic command block starts with the input line

```
BEGIN PARAMETERS FOR MODEL ELASTIC
```

and terminates with the input line

```
END [PARAMETERS FOR MODEL ELASTIC] .
```

In the above command blocks:

- Density is defined with the `DENSITY` command line.
- Young's modulus is defined with the `YOUNGS MODULUS` command line.
- Poisson's ratio is defined with the `POISSONS RATIO` command line.
- The bulk modulus is defined with the `BULK MODULUS` command line.
- The shear modulus is defined with the `SHEAR MODULUS` command line.
- Lambda is defined with the `LAMBDA` command line.

Only two of the elastic constants are required.

### 2.2.2 Elastic-Plastic Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ELASTIC_PLASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING MODULUS = <real>hardening_modulus
    BETA = <real>beta_parameter(1.0)
  END [PARAMETERS FOR MODEL ELASTIC_PLASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The elastic-plastic linear hardening models are used to model materials, generally metals, undergoing plastic deformation at finite strains. Linear hardening generally refers to the shape of a uniaxial stress-strain curve where the stress increases linearly with the plastic, or permanent, strain. In a three-dimensional framework, hardening is the law that governs how the yield surface grows in stress space. If the yield surface grows uniformly in stress space, the hardening is referred to as isotropic hardening. When `BETA` is 1.0, we have only isotropic hardening.

Because the linear hardening model is relatively simple to integrate, there is also the ability to define a yield surface that not only grows, or hardens, but also moves in stress space. This is known as kinematic hardening. When `BETA` is 0.0, we have only kinematic hardening. The elastic-plastic linear hardening model allows for isotropic hardening, kinematic hardening, or a combination of the two.

For an elastic-plastic material, an elastic-plastic command block starts with the input line

```
BEGIN PARAMETERS FOR MODEL ELASTIC_PLASTIC
```

and terminates with the input line

```
END [PARAMETERS FOR MODEL ELASTIC_PLASTIC] .
```

In the above command blocks:

- Density is defined with the `DENSITY` command line.
- Young's modulus is defined with the `YOUNGS MODULUS` command line.
- Poisson's ratio is defined with the `POISSONS RATIO` command line.
- The bulk modulus is defined with the `BULK MODULUS` command line.
- The shear modulus is defined with the `SHEAR MODULUS` command line.
- Lambda is defined with the `LAMBDA` command line.
- The yield stress is defined with the `YIELD STRESS` command line.
- The hardening modulus is defined with the `HARDENING MODULUS` command line.
- The beta parameter is defined with the `BETA` command line.

Only two of the elastic constants are required.

### 2.2.3 Elastic-Plastic Power-Law Hardening Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL EP_POWER_HARD
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING CONSTANT = <real>hardening_constant
    HARDENING EXPONENT = <real>hardening_exponent
    LUDERS STRAIN = <real>luders_strain
  END [PARAMETERS FOR MODEL EP_POWER_HARD]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

A power-law hardening model for elastic-plastic materials is used for modeling metal plasticity up to finite strains. The power-law hardening model, as opposed to the linear hardening model, has a power law fit for the uniaxial stress-strain curve that has the stress increase as the plastic strain raised to a power. The power-law hardening model also has the ability to model materials that exhibit Luder's strains after yield. Due to the more complicated yield behavior, the power-law hardening model can only be used with isotropic hardening.

For an elastic-plastic power-law hardening material, an elastic-plastic power-law hardening command block starts with the input line

```
BEGIN PARAMETERS FOR MODEL EP_POWER_HARD
```

and terminates with the input line

```
END [PARAMETERS FOR MODEL EP_POWER_HARD] .
```

In the above command blocks:

- Density is defined with the `DENSITY` command line.
- Young's modulus is defined with the `YOUNGS MODULUS` command line.
- Poisson's ratio is defined with the `POISSONS RATIO` command line.
- The bulk modulus is defined with the `BULK MODULUS` command line.
- The shear modulus is defined with the `SHEAR MODULUS` command line.
- Lambda is defined with the `LAMBDA` command line.
- The yield stress is defined with the `YIELD STRESS` command line.
- The hardening constant is defined with the `HARDENING CONSTANT` command line.
- The hardening exponent is defined with the `HARDENING EXPONENT` command line.
- The Luder's strain is defined with the `LUDERS STRAIN` command line.

Only two of the elastic constants are required.

### 2.2.4 Soil and Crushable Foam Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL SOIL_FOAM
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    A0 = <real>const_coeff_yieldsurf
    A1 = <real>lin_coeff_yieldsurf
    A2 = <real>quad_coeff_yieldsurf
    PRESSURE CUTOFF = <real>pressure_cutoff
    PRESSURE FUNCTION = <string>function_press_volstrain
  END [PARAMETERS FOR MODEL SOIL_FOAM]
END [PROPERTY SPECIFICIATION FOR MATERIAL <string>mat_name]
```

The soil and crushable foam model is a plasticity model that can be used for modeling soil or crushable foam. Given the right input, the model is a Drucker-Prager model.

For the soil and crushable foam model, the yield surface is a surface of revolution about the hydrostat in principal stress space. A planar end cap is assumed for the yield surface so that the yield surface is closed. The yield stress, $\sigma_{yd}$, is specified as a polynomial in pressure, $p$. The yield stress is given as

$$\sigma_{yd} = a_0 + a_1 p + a_2 p, \tag{1}$$

where $p$ is positive in compression. The determination of the yield stress from Equation (1) places severe restrictions on the admissible values of $a_0$, $a_1$, and $a_2$. There are three valid cases for the yield surface. In the first case, there is an elastic-perfectly plastic deviatoric response, and the yield surface is a cylinder oriented along the hydrostat in principal stress space. In this case, $a_0$ is positive, and $a_1$ and $a_2$ are zero. In the second case, the yield surface is conical. A conical yield surface is obtained by setting $a_2$ to zero and using appropriate values for $a_0$ and $a_1$. In the third case, the yield surface has a parabolic shape. For the parabolic yield surface, all three of the coefficients in Equation (1) are nonzero. The coefficients are checked to determine that a valid negative tensile-failure pressure can be derived based on the specified coefficients.

For the case of the cylindrical yield surface (e.g. $a_0 > 0$ and $a_1 = a_2 = 0$), there is no tensile-failure pressure. For the other two cases, the computed tensile-failure pressure may be too low. To handle the situations where there is no tensile-failure pressure or the tensile-failure pressure is too low, a pressure cutoff can be defined. If a pressure cutoff is defined, the tensile-failure pressure is the larger of the computed tensile-failure pressure and the defined cutoff pressure.

The plasticity theories for the volumetric and deviatoric parts of the material response are completely uncoupled. The volumetric response is computed first. The mean pressure $p$ is assumed to be positive in compression, and a yield function $\phi_p$ is written for the volumetric response as

$$\phi_p = p - f_p(\varepsilon_V), \tag{2}$$

where $f_p(\varepsilon_V)$ defines the volumetric stress-strain curve for the pressure. The yield function $\phi_p$ determines the motion of the end cap along the hydrostat.

For a soil and crushable foam material, a soil and crushable foam command block starts with the input line

```
BEGIN PARAMETERS FOR MODEL SOIL_FOAM
```

and terminates with the input line

```
END [PARAMETERS FOR MODEL SOIL_FOAM] .
```

In the above command blocks:

- Density is defined with the `DENSITY` command line.
- Young's modulus is defined with the `YOUNGS MODULUS` command line.
- Poisson's ratio is defined with the `POISSONS RATIO` command line.
- The bulk modulus is defined with the `BULK MODULUS` command line.

- The shear modulus is defined with the `SHEAR MODULUS` command line.
- Lambda is defined with the `LAMBDA` command line.
- The constant in the equation for the yield surface is defined with the `A0` command line.
- The coefficient for the linear term in the equation for the yield surface is defined with the `A1` command line.
- The coefficient for the quadratic term in the equation for the yield surface is defined with the `A2` command line.
- The user-defined tensile-failure pressure is defined with the `PRESSURE CUTOFF` command line.
- The pressure as a function of volumetric strain is defined with the function named on the `PRESSURE FUNCTION` command line.

Only two of the elastic constants are required.

### 2.2.5 Foam Plasticity Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL FOAM_PLASTICITY
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    PHI = <real>phi
    SHEAR STRENGTH  = <real>shear_strength
    SHEAR HARDENING = <real>shear_hardening
    SHEAR EXPONENT  = <real>shear_exponent
    HYDRO STRENGTH  = <real>hydro_strength
    HYDRO HARDENING = <real>hydro_hardening
    HYDRO EXPONENT  = <real>hydro_exponent
    BETA = <real>beta
  END [PARAMETERS FOR MODEL FOAM_PLASTICITY]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The foam plasticity model was developed to describe the response of porous elastic-plastic materials like closed-cell polyurethane foam to large deformation. Like solid metals, these foams can exhibit significant plastic deviatoric strains (permanent shape changes). Unlike metals, these foams can also exhibit significant plastic volume strains (permanent volume changes). The foam plasticity model is characterized by an initial yield surface that is an ellipsoid about the hydrostat.

When foams are compressed, they typically exhibit an initial elastic regime followed by a plateau regime in which the stress needed to compress the foam remains nearly constant. At some point in the compression process the densification regime is reached, and the stress needed to compress the foam further begins to rapidly increase.

The foam plasticity model can be used to describe the response of metal foams and many closed-cell, polymeric foams to large deformation (including polyurethane, polystyrene bead, etc.). This model is not appropriate for flexible foams that return to their undeformed shape after loads are removed.

For a foam plasticity material, a foam plasticity command block starts with the input line

```
BEGIN PARAMETERS FOR MODEL FOAM_PLASTICITY
```

and terminates with the input line

```
END [PARAMETERS FOR MODEL FOAM_PLASTICITY] .
```

In the above command blocks:

- Density is defined with the `DENSITY` command line.
- Young's modulus is defined with the `YOUNGS MODULUS` command line.
- Poisson's ratio is defined with the `POISSONS RATIO` command line.
- The bulk modulus is defined with the `BULK MODULUS` command line.
- The shear modulus is defined with the `SHEAR MODULUS` command line.
- Lambda is defined with the `LAMBDA` command line.
- The initial volume fraction of solid material in the foam, $\varphi$, is defined with the `PHI` command line. For example, solid polyurethane weighs 75 pounds per cubic foot (pcf); uncompressed 10 pcf polyurethane foam would have a $\varphi$ of $0.133 = 10/75$.
- The shear (deviatoric) strength of uncompressed foam is defined with the `SHEAR STRENGTH` command line.
- The shear hardening modulus for the foam is defined with the `SHEAR HARDENING` command line.
- The shear hardening exponent is defined with the `SHEAR EXPONENT` command line. The deviatoric strength is given by `SHEAR_STRENGTH + SHEAR_HARDENING * PHI**SHEAR_EXPONENT.`
- The hydrostatic (volumetric) strength of the uncompressed foam is defined with the `HYDRO STRENGTH` command line.
- The hydrodynamic hardening modulus for the foam is defined with the `HYDRO HARDENING` command line.
- The hydrodynamic hardening exponent for the foam is defined with the `HYDRO EXPONENT` command line. The hydrostatic strength is given by `HYDRO_STRENGTH + HYDRO_HARDENING * PHI**HYDRO_EXPONENT.`
- The prescription for nonassociated flow, $\beta$, is defined with the `BETA` command line. When $\beta = 0.0$, the flow direction is given by the normal to the yield surface (associated flow). When $\beta = 1.0$, the flow direction is given by the stress tensor. Values of $\beta$ between 0.0 and 0.95 are recommended.

Only two of the elastic constants are required.

### 2.2.6 Orthotropic Crush Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    EX  = <real>modulus_x
    EY  = <real>modulus_y
    EZ  = <real>modulus_z
    GXY = <real>shear_modulus_xy
    GYZ = <real>shear_modulus_yz
    GZX = <real>shear_modulus_zx
    VMIN = <real>min_crush_volume
    CRUSH XX = <string>stress_volume_xx_function_name
    CRUSH YY = <string>stress_volume_yy_function_name
    CRUSH ZZ = <string>stress_volume_zz_function_name
    CRUSH XY = <string>shear_stress_volume_xy_function_name
    CRUSH YZ = <string>shear_stress_volume_yz_function_name
    CRUSH ZX = <string>shear_stress_volume_zx_function_name
  END [PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The orthotropic crush model is an empirically based constitutive relation that is useful for modeling materials like metallic honeycomb and wood. This particular implementation follows the formulation of the metallic honeycomb model in DYNA3D [6]. The orthotropic crush model divides material behavior into three phases:

- orthotropic elastic,
- volumetric crush (partially compacted), and
- elastic-perfectly plastic (fully compacted).

For an orthotropic crush material, an orthotropic crush command block starts with the input line

```
BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
```

and terminates with the input line

```
END [PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH] .
```

In the above command blocks:

- The uncompacted density is defined with the `DENSITY` command line.
- Young's modulus for the fully compacted state is defined with the `YOUNGS MODULUS` command line. This is the elastic-perfectly plastic value of Young's modulus.

- Poisson's ratio for the fully compacted state is defined with the `POISSONS RATIO` command line. This is the elastic-perfectly plastic value of Poisson's ratio.
- The bulk modulus is defined with the `BULK MODULUS` command line.
- The shear modulus is defined with the `SHEAR MODULUS` command line.
- Lambda is defined with the `LAMBDA` command line.
- The yield stress for the fully compacted state is defined with the `YIELD STRESS` command line. This is the elastic-perfectly plastic value of the yield stress.
- The initial directional modulus $E_{xx}$ is defined with the `EX` command line.
- The initial directional modulus $E_{yy}$ is defined with the `EY` command line.
- The initial directional modulus $E_{zz}$ is defined with the `EZ` command line.
- The initial directional shear modulus $G_{xy}$ is defined with the `EXY` command line.
- The initial directional shear modulus $G_{yz}$ is defined with the `EYZ` command line.
- The initial directional shear modulus $G_{zx}$ is defined with the `EZX` command line.
- The minimum crush volume as a fraction of the original volume is defined with the `VMIN` command line.
- The directional stress $\sigma_{xx}$ as a function of the volume crush is defined by the function referenced in the `CRUSH XX` command line.
- The directional stress $\sigma_{yy}$ as a function of the volume crush is defined by the function referenced in the `CRUSH YY` command line.
- The directional stress $\sigma_{zz}$ as a function of the volume crush is defined by the function referenced in the `CRUSH ZZ` command line.
- The directional stress $\sigma_{xy}$ as a function of the volume crush is defined by the function referenced in the `CRUSH XY` command line.
- The directional stress $\sigma_{yz}$ as a function of the volume crush is defined by the function referenced in the `CRUSH YZ` command line.
- The directional stress $\sigma_{zx}$ as a function of the volume crush is defined by the function referenced in the `CRUSH ZX` command line.

Only two of the elastic constants are required. Note that several of the command lines in this command block (those beginning with `CRUSH`) reference functions. These functions must be defined in the domain scope.

### 2.2.7 Orthotropic Rate Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_RATE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    MODULUS TTTT = <real>modulus_tttt
    MODULUS TTLL = <real>modulus_ttll
    MODULUS TTWW = <real>modulus_ttww
    MODULUS LLLL = <real>modulus_llll
    MODULUS LLWW = <real>modulus_llww
    MODULUS WWWW = <real>modulus_wwww
    MODULUS TLTL = <real>modulus_tltl
    MODULUS LWLW = <real>modulus_lwlw
    MODULUS WTWT = <real>modulus_wtwt
    TX = <real>tx
    TY = <real>ty
    TZ = <real>tz
    LX = <real>lx
    LY = <real>ly
    LZ = <real>lz
    MODULUS FUNCTION = <string>modulus_function_name
    RATE FUNCTION = <string>rate_function_name
    T FUNCTION = <string>t_function_name
    L FUNCTION = <string>l_function_name
    W FUNCTION = <string>w_function_name
    TL FUNCTION = <string>tl_function_name
    LW FUNCTION = <string>lw_function_name
    WT FUNCTION = <string>wt_function_name
  END [PARAMETERS FOR MODEL ORTHOTROPIC_RATE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

Orthotropic rate is a new and improved version of the orthotropic crush constitutive model. This model has been developed to describe the behavior of an aluminum honeycomb subjected to large deformation. The new orthotropic rate model, like the original orthotropic crush model, has six independent yield functions that evolve with volume strain. Unlike the original model, this new model has yield functions that also depend on strain rate. The new model also uses an orthotropic elasticity tensor with nine elastic moduli in place of the orthotropic elasticity tensor with six elastic moduli used in the original orthotropic crush model. A new honeycomb orientation capability has also been added that allows users to prescribe initial honeycomb orientations that are not aligned with the original global coordinate system.

For an orthotropic rate material, an orthotropic rate command block starts with the input line

```
BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_RATE
```

and terminates with the input line

```
END [PARAMETERS FOR MODEL ORTHOTROPIC_RATE] .
```

In the above command blocks:

- Density is defined with the `DENSITY` command line.
- Young's modulus for the fully compacted honeycomb is defined with the `YOUNGS MODULUS` command line.
- The yield stress for the fully compacted honeycomb is defined with the `YIELD STRESS` command line.
- The nine elastic moduli for the orthotropic uncompacted honeycomb are defined with the `MODULUS TTT`, `MODULUS TTLL`, `MODULUS TTWW`, `MODULUS LLLL`, `MODULUS LLWW`, `MODULUS WWWW`, `MODULUS TLTL`, `MODULUS LWLW`, and `MODULUS WTWT` command lines. The T-direction is usually associated with the generator axis for the honeycomb. The L-direction is in the ribbon plane (plane defined by flat sheets used in reinforced honeycomb) and orthogonal to the generator axis. The W-direction is perpendicular to the ribbon plane.
- The components of a vector defining the T-direction of the honeycomb are defined by the `TX`, `TY`, and `TZ` command lines. The values `tx`, `ty`, and `tz` are components of a vector in the global coordinate system that define the orientation of the honeycomb's T-direction (generator axis).
- The components of a vector defining the L-direction of the honeycomb are defined by the `LX`, `LY`, and `LZ` command lines. The values `lx`, `ly`, and `lz` are components of a vector in the global coordinate system that define the orientation of the honeycomb's L-direction. *Caution*: The vectors T and L must be orthogonal.
- The function describing the variation in moduli with compaction is given by the `MODULUS FUNCTION` command line. The moduli vary continuously from their initial orthotropic values to isotropic values when full compaction is obtained.
- The function describing the change in strength with strain rate is given by the `RATE FUNCTION` command line. Note that all strengths are scaled with the multiplier obtained from this function.
- The function describing the T-normal strength of the honeycomb as a function of compaction is given by the `T FUNCTION` command line.
- The function describing the L-normal strength of the honeycomb as a function of compaction is given by the `L FUNCTION` command line.
- The function describing the W-normal strength of the honeycomb as a function of compaction is given by the `W FUNCTION` command line.
- The function describing the TL-normal strength of the honeycomb as a function of compaction is given by the `TL FUNCTION` command line.
- The function describing the LW-normal strength of the honeycomb as a function of compaction is given by the `LW FUNCTION` command line.
- The function describing the WT-normal strength of the honeycomb as a function of compaction is given by the `WT FUNCTION` command line.

Only the elastic modulus (Young's modulus) is required for this model. If two elastic constants are supplied, the elastic constants will be completed. However, only the elastic modulus is used in this model. Note that several of the command lines in this command block reference functions. These functions must be defined in the domain scope.

### 2.2.8 Mie-Gruneisen Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN
    RHO_0 = <real>density
    C_0 = <real>sound_speed
    SHUG = <real>const_shock_velocity
    GAMMA_0 = <real>ambient_gruneisen_param
    POISSR = <real>poissons_ratio
    Y_0 = <real>yield_strength
    PMIN = <real>mean_stress(REAL_MAX)
  END [PARAMETERS FOR MODEL MIE_GRUNEISEN]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Mie-Gruneisen material model describes the nonlinear pressure-volume (or equivalently pressure-density) response of solids or fluids in terms of a reference pressure-volume curve and deviations from the reference curve in energy space. The reference curve is taken to be the experimentally determined principal Hugoniot, which is the locus of end states that can be reached by a shock transition from the ambient state. For details about this model, see Reference 7.

For Mie-Gruneisen energy-dependent materials, the Mie-Gruneisen command block begins with the input line

```
BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN
```

and is terminated with the input line

```
END [PARAMETERS FOR MODEL MIE_GRUNEISEN] .
```

In the above command blocks:

- The ambient density, $\rho_0$, is defined with the `RHO_0` command line. The ambient density is the density at which the mean pressure is zero, not necessarily the initial density.
- The ambient bulk sound speed, $c_0$, is defined by the `C_0` command line. The ambient bulk sound speed is also the first constant in the shock-velocity-versus-particle-velocity relation $D = c_0 + Su$. (See the following description of the `SHUG` command line for the definition of *S*.)
- The second constant in the shock-velocity-versus-particle-velocity equation, *S*, is defined by the in the `SHUG` command line. The shock-velocity-versus-particle-velocity relation is $D = c_0 + Su$. (See the previous description of the `C_0` command line for the definition of $c_0$.)

- The ambient gruneisen parameter, $\Gamma_0$, is defined by the `GAMMA_0` command line.
- Poisson's ratio, $\nu$, is defined by the `POISSR` command line. Poisson's ratio is assumed constant.
- The yield strength, $y_0$, is defined by the `Y_0` command line. The yield strength is zero for the hydrodynamic case.
- The fracture stress is defined by the `PMIN` command line. The fracture stress is a mean stress or pressure, so it must be negative or zero. This is an optional parameter; if not specified, the parameter defaults to `REAL_MAX` (no fracture).

### 2.2.9 Mie-Gruneisen Power-Series Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES
    RHO_0 = <real>density
    C_0 = <real>sound_speed
    K1 = <real>power_series_coeff1
    K2 = <real>power_series_coeff2
    K3 = <real>power_series_coeff3
    K4 = <real>power_series_coeff4
    K5 = <real>power_series_coeff5
    GAMMA_0 = <real>ambient_gruneisen_param
    POISSR = <real>poissons_ratio
    Y_0 = <real>yield strength
    PMIN = <real>mean_stress(REAL_MAX)
  END [PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Mie-Gruneisen power-series model describes the nonlinear pressure-volume (or equivalently pressure-density) response of solids or fluids in terms of a reference pressure-volume curve and deviations from the reference curve in energy space. The reference curve is taken to be the experimentally determined principal Hugoniot, which is the locus of end states that can be reached by a shock transition from the ambient state. The Mie-Gruneisen power-series model is very similar to the Mie-Gruneisen model, except that the Mie-Gruneisen model bases the Hugoniot pressure-volume response on the assumption of a linear shock-velocity-versus-particle-velocity relation, while the Mie-Gruneisen power-series model uses a power-series expression. For details about this model, see Reference 7.

For Mie-Gruneisen power-series energy-dependent materials, the Mie-Gruneisen power-series command block begins with the input line

```
BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES
```

and is terminated with the input line

```
END [PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES] .
```

In the above command blocks:

- The ambient density, $\rho_0$, is defined with the `RHO_0` command line. The ambient density is the density at which the mean pressure is zero, not necessarily the initial density.
- The ambient bulk sound speed, $c_0$, is defined by the `C_0` command line.
- The power-series coefficients $k_1$, $k_2$, $k_3$, $k_4$, and $k_5$ are defined by the command lines `K1`, `K2`, `K3`, `K4`, and `K5`, respectively. Only the nonzero power-series coefficients need be input, since coefficients not specified will default to zero.
- The ambient gruneisen parameter, $\Gamma_0$, is defined by the `GAMMA_0` command line.
- Poisson's ratio, $\nu$, is defined by the `POISSR` command line. Poisson's ratio is assumed constant.
- The yield strength, $y_0$, is defined by the `Y_0` command line. The yield strength is zero for the hydrodynamic case.
- The fracture stress is defined by the `PMIN` command line. The fracture stress is a mean stress or pressure, so it must be negative or zero. This is an optional parameter; if not specified, the parameter defaults to `REAL_MAX` (no fracture).

### 2.2.10 JWL (Jones-Wilkins-Lee) Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL JWL
    RHO_0 = <real>initial_density
    D = <real>detonation_velocity
    E_0 = <real>init_chem_energy
    A = <real>jwl_const_pressure1
    B = <real>jwl_const_pressure2
    R1 = <real>jwl_const_nondim1
    R2 = <real>jwl_const_nondim2
    OMEGA = <real>jwl_const_nondim3
    XDET = <real>x_detonation_point
    YDET = <real>y_detonation_point
    ZDET = <real>z_detonation_point
    TDET = <real>time_of_detonation
    B5 = <real>burn_width_const(2.5)
  END [PARAMETERS FOR MODEL JWL]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The JWL model describes the pressure-volume-energy response of the gaseous detonation products of HE (High Explosive). For details about this model, see Reference 7.

For JWL energy-dependent materials, the JWL command block begins with the input line

```
BEGIN PARAMETERS FOR MODEL JWL
```

and is terminated with the input line

```
END [PARAMETERS FOR MODEL JWL] .
```

In the above command blocks:

- The initial density of the unburned explosive, $\rho_0$, is given by the `RHO_0` command line.
- The detonation velocity, *D*, is given by the `D` command line.
- The initial chemical energy per unit mass in the explosive, $E_0$, is given by the `E_0` command line. Most compilations of JWL parameters give $E_0$ in units of energy per unit volume, rather than energy per unit mass. Thus, the tabulated value must be divided by $\rho_0$, the initial density of the unburned explosive.
- The JWL constants with units of pressure, *A* and *B*, are given by the `A` and `B` command lines, respectively.
- The dimensionless JWL constants, $R_1$, $R_2$, and $\omega$, are given by the `R1`, `R2`, and `OMEGA` command lines, respectively.
- The *x*-coordinate of the detonation point, $x_D$, is given by the `XDET` command line.
- The *y*-coordinate of the detonation point, $y_D$, is given by the `YDET` command line.
- The *z*-coordinate of the detonation point, $z_D$, is given by the `ZDET` command line.
- The time of detonation, $t_D$, is given by the `TDET` command line.
- The burn-width constant, $B_5$, is given by the `B5` command line. The burn-width constant has a default value of 2.5.

### 2.2.11 Ideal Gas Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL IDEAL_GAS
    RHO_0 = <real>initial_density
    C_0 = <real>initial_sound_speed
    GAMMA = <real>ratio_specific_heats
  END [PARAMETERS FOR MODEL IDEAL_GAS]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The ideal gas model provides a material description based on the ideal gas law. For details about this model, see Reference 7.

For ideal gas materials, the ideal gas command block begins with the input line

```
BEGIN PARAMETERS FOR MODEL IDEAL_GAS
```

and is terminated with the input line

```
END [PARAMETERS FOR MODEL IDEAL_GAS] .
```

In the above command blocks:

- The initial density, $\rho_0$, is given by the `RHO_0` command line.
- The initial sound speed, $c_0$, is given by the `C_0` command line.
- The ratio of specific heats, $\gamma$, is given by the `GAMMA` command line.

## 2.3 Finite Element Model

```
BEGIN FINITE ELEMENT MODEL <string>mesh_descriptor
  DATABASE NAME = <string>mesh_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  ALIAS <string>mesh_identifier AS <string>user_name .
  BEGIN PARAMETERS FOR BLOCK <string>block_name
    #
    # Command lines that define attributes for
    # a particular element block appear in this
    # command block.
    #
  END [PARAMETERS FOR BLOCK <string>block_name]
END [FINITE ELEMENT MODEL <string>mesh_descriptor]
```

The Presto input file must contain a description of the mesh file that is to be used for an analysis. The mesh file is described by a `FINITE ELEMENT MODEL` command block that appears in the domain scope.

The command block to describe a mesh file begins with

```
BEGIN FINITE ELEMENT MODEL <string>mesh_descriptor
```

and is terminated with

```
END FINITE ELEMENT MODEL <string>mesh_descriptor ,
```

where `mesh_descriptor` is a user-selected name for the mesh.

Nested within the command block are two command lines (`DATABASE NAME` and `DATABASE TYPE`) that give the mesh name and define the type for the mesh file, respectively. The command line

```
DATABASE NAME = <string>mesh_file_name ,
```

gives the name of the mesh file with the string `mesh_file_name`. If the current mesh file is in the default directory with the input file and is named `job.g`, then this command line would appear as

```
DATABASE NAME = job.g .
```

If the mesh file is in some other directory, the command line would have to show the path to that directory. For parallel runs, the string `mesh_file_name` is the base name for the spread of parallel mesh files. For example, for a four-processor run, the actual mesh files associated with a base name of `job.g` would be `job.g.4.0`, `job.g.4.1`, `job.g.4.2`, and `job.g.4.3`. The database name on the command line would be `job.g`.

If the mesh file does not use the Exodus II format, you must specify the format for the mesh file using the command line

```
DATABASE TYPE = <string>database_type(exodusII) .
```

Currently, only the Exodus II database is supported by the SIERRA Framework. Other options will be added in the future.

It is possible to associate a user-defined name with some mesh entity. The mesh entity for Exodus II relies on some type of integer identification. You can relate the integer identification to some name that is more descriptive by using the `ALIAS` command line:

```
ALIAS <string>mesh_identifier AS <string>user_name .
```

Examples of this association are as follows:

```
Alias block_1   as Case
Alias block_10  as Fin
Alias block_12  as Nose
Alias surface_1 as Nose_Case_Interface
Alias surface_2 as OuterBoundary
```

The above examples use the Exodus II naming convention described in Section 1.4.

Within the `FINITE ELEMENT MODEL` command block, there will be one or more `PARAMETERS FOR BLOCK` command blocks, which are also referred to in this document as *element-block command blocks*. The finite element model consists of one or more element blocks. The basic information about the element blocks (number of elements, topology, connectivity, etc.) is read from a mesh file. The specific element type to be used by Presto is not in the mesh file. For example, the mesh file may specify a topology such as a four-node quadrilateral for an element block. If a command line appears in the command block for this element block that specifies a membrane thickness, then the element type will be a four-node membrane element. If, on the other hand, a command line appears in the command block for this element block that specifies a shell thickness, then the element type will be a four-node shell element. Line commands in the element-block command block establish the specific element type as opposed to the basic element topology.

All of the elements within an element-block command block will have the same mechanics properties. The mechanics properties are set by use of the various command lines. For example, there is a command line that will let you define the material properties for the elements in the block. There is a command line that will let you specify hourglass control parameters for the elements in the block (if these elements use hourglass control). This command line will overwrite the default hourglass control parameters, and they will apply only to the elements of this particular element block. All of the command lines that can be used for the element-block command block are described in Section 2.3.1 through Section 2.3.11.

The element-block command block begins with the input line

```
BEGIN PARAMETERS FOR BLOCK <string>block_name
```

and is terminated with the input line

```
END [PARAMETERS FOR BLOCK <string>block_name] ,
```

where `block_name` is the name assigned to the element block. If the format for the mesh file is Exodus II, the typical form of `block_name` is `block_integerid`, where `integerid` is the integer identifier for the block. If the element block is 280, the value of

block_name would be block_280. It is also possible to generate an alias identifier for the element block and use this for the block_name. If block_280 is aliased to AL6061, then block_name becomes AL6061.

Currently, the elements supported in Presto are as follows:

- Eight-node, uniform-gradient hexahedron: Both a midpoint-increment formulation [8] and a strongly objective formulation are implemented [11].
- Four-node tetrahedron: Only a strongly objective formulation is implemented.
- Eight-node tetrahedron: This tetrahedral element has nodes at the four vertices and nodes on the four faces. The eight-node tetrahedron has only a strongly objective formulation [12].
- Ten-node tetrahedron: Only a strongly objective formulation is implemented.
- Four-node quadrilateral membrane: Both a midpoint-increment formulation and a strongly objective formulation are implemented. This element is derived from the Key-Hoff shell formulation [13].
- Four-node quadrilateral shell: This shell uses the Key-Hoff formulation [13]. Both a midpoint-increment formulation and a strongly objective formulation are implemented.
- Two-node truss: The two-node truss element carries only a uniform axial stress at present. This element uses only a linear-elastic material model at present.
- Two-node damper: The two-node damping element computes a damping force based on the relative velocity of the two nodes along the axis of the element.

### 2.3.1 Definition of Material Model

```
MATERIAL <string>material_name
SOLID MECHANICS USE MODEL <string>model_name
```

The property specification for an element block is done by using the above two command lines. The property specification references both a PROPERTY SPECIFICATION FOR MATERIAL command block and a material-model command block, which has the general form PARAMETERS FOR MODEL model_name . These command blocks are described in Section 2.2. The PROPERTY SPECIFICATION FOR MATERIAL command block contains all of the parameters needed to define a material, and is associated with an element block (PARAMETERS FOR BLOCK command block) by use of the MATERIAL command line. Some of the material parameters inside the property specification are grouped on the basis of material models. A material-model command block is associated with an element block by use of the SOLID MECHANICS USE MODEL command line.

Consider the following example. Suppose there is a PROPERTY SPECIFICATION FOR MATERIAL command block with a material_name of steel. Embedded within this command block for steel is a material-model command block for an elastic model of steel and an elastic-plastic model of steel. Suppose that for the current element block

we would like to use the material `steel` with the elastic model. Then the element-block command block would contain the input lines

```
MATERIAL steel
SOLID MECHANICS USE MODEL elastic .
```

If, on the other hand, we would like to use the material `steel` with the elastic-plastic model, the element-block command block would contain the input lines

```
MATERIAL steel
SOLID MECHANICS USE MODEL elastic_plastic .
```

The user should remember that not all material types can be used with all element types.

### 2.3.2 Element Strain Formulation

```
ELEMENT STRAIN FORMULATION = <string>midpoint|
  strongly-objective(midpoint)
```

Using the `ELEMENT STRAIN FORMULATION` command line, you can specify either a midpoint-increment strain formulation (`midpoint`) or a strongly objective strain formulation (`strongly-objective`) for some of the elements. See the general element description to determine which elements offer both formulations. Consult the element documentation [11,14] for a description of these two strain formulations.

### 2.3.3 Linear and Quadratic Bulk Viscosity

```
LINEAR BULK VISCOSITY =
<real>linear_bulk_viscosity_value(0.06)
QUADRATIC BULK VISCOSITY =
<real>quad_bulk_viscosity_value(1.20)
```

The linear and quadratic bulk viscosity are set with these two command lines. Consult the documentation for the elements [14] for a description of the bulk viscosity parameters.

### 2.3.4 Hourglass Control

```
HEX HOURGLASS STIFFNESS = <real>hour_glass_stiff_value(0.05)
HEX HOURGLASS VISCOSITY = <real>hour_glass_visc_value(0.0)
```

The hourglass control for uniform-gradient, eight-node hexahedral elements is set with these two command lines. Consult the element documentation [14] for a description of the hourglass parameters.

The hourglass stiffness is the same as the dilatational hourglass parameter, and the hourglass viscosity is the same as the deviatoric hourglass parameter.

### 2.3.5 Membrane Scale Thickness

```
MEMBRANE SCALE THICKNESS = <real>mem_scale_thick_value
```

This command line scales the thickness associated with each membrane element in the mesh file. Currently, the SIERRA Framework input-output (IO) routines specify a thickness of 1.0 for all elements with a four-node quadrilateral topology. If the value for `mem_scale_thick_value` is set to 0.25, the thickness of the membrane elements in the element block will be 0.25. Future enhancements to the SIERRA Framework IO routines will let the user read in a unique thickness for each four-node quadrilateral element in a block of four-node quadrilateral elements.

If the `MEMBRANE SCALE THICKNESS` command line appears in an element-block command block, then the elements in the block must have a four-node quadrilateral topology.

### 2.3.6 Control Parameters for Shell Elements

```
SHELL SCALE THICKNESS = <real>shell_scale_thick_value
SHELL INTEGRATION POINTS =
  <integer>number_integration_points(5)
SHELL INTEGRATION SCHEME = <string>GAUSS|LOBATTO|
  TRAPEZOID(TRAPEZOID)
```

The above three command lines all set parameters for shell elements. The command line

```
SHELL SCALE THICKNESS = <real>shell_scale_thick_value
```

scales the thickness associated with each shell element in the mesh file. Currently, the SIERRA Framework IO routines specify a thickness of 1.0 for all elements with a four-node quadrilateral topology. If the value for `shell_scale_thick_value` is set to 0.25, the thickness of the shell elements in the element block will be 0.25. Future enhancements to the SIERRA Framework IO routines will let the user read in a unique thickness for each four-node quadrilateral element in a block of four-node quadrilateral elements.

The command line

```
SHELL INTEGRATION POINTS =
  <integer>number_integration_points(5)
```

specifies the number of integration points to be used through the thickness of the shell. The actual integration scheme is selected by using the command line

```
SHELL INTEGRATION SCHEME = <string>GAUSS|LOBATTO|
  TRAPEZOID(TRAPEZOID) .
```

Currently, **only** the trapezoid integration scheme with five integration points is implemented. Consult the element documentation [14] for a description of different integration schemes for shell elements.

If the command lines for shell elements appear in an element-block command block, then the elements in the block must have a four-node quadrilateral topology.

### 2.3.7 Truss Area

```
TRUSS AREA = <real>truss_cross_sectional_area
```

The cross-sectional area for truss elements is specified by the `TRUSS AREA` command line. The value `truss_cross_sectional_area` is the cross-sectional area of the truss members in the element block. If this command line is used for a block of three-dimensional, two-node elements, the elements in the block are treated as truss elements.

### 2.3.8 Damper Area

```
DAMPER AREA = <real>damper_cross_sectional_area
```

The cross-sectional area for damper elements is specified by the `DAMPER AREA` command line. The value `damper_cross_sectional_area` is the cross-sectional area of the dampers in the element block. If this command line is used for a block of three-dimensional, two-node elements, the elements in the block are treated as damper elements.

The damper area is used only to generate mass associated with the damper element. The mass is the density for the damper element multiplied by the original volume of the element (original length multiplied by the damper area).

The force generated by the damper element depends on the relative velocity along the current direction vector for the damper element. If $\boldsymbol{n}$ is a unit normal pointing in the direction from node 1 to node 2 and $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are the velocity vectors at nodes 1 and 2, respectively, then the force generated by the damper element is

$$F_d = \eta \boldsymbol{n} \cdot (\boldsymbol{v}_2 - \boldsymbol{v}_1), \tag{3}$$

where $\eta$ is the damping parameter. Currently, the damping parameter must be specified by using an elastic material model for the damper element. The value for Young's modulus in the elastic material model is used for the damping parameter $\eta$.

### 2.3.9 Energy Deposition

```
DEPOSIT SPECIFIC INTERNAL ENERGY <real>edep [OVER TIME
<real>tdep STARTING AT TIME <real>tinit]
```

This command line controls the amount of energy deposited from external sources, *dQ*. The command line defines the amount, `edep`, of specific (per unit mass) internal energy to be deposited in the material. The energy is deposited over time, `tdep`, beginning at time `tinit`. The optional parameters `tdep` and `tinit` both default to zero, so the energy will be deposited instantaneously at time zero if they are not specified. The deposition is uniform in space, so each element in the block has the same amount, `edep`, added to its specific internal energy.

This command is applicable only to the energy-dependent material models (Mie-Gruneisen, Mie-Gruneisen Power-Series, JWL, Ideal Gas), which are described in Reference 7.

### 2.3.10 Element Numerical Formulation

```
ELEMENT NUMERICAL FORMULATION = old | new (old)
```

For calculation of the critical time step, it is necessary to determine a characteristic length for each element. In one dimension the correct characteristic element length is obviously the distance between the two nodes of the element. In higher dimensions this is usually taken to be the minimum distance between any of the nodes in the element. However, some finite element codes, primarily those based on Pronto3D [8], use as a characteristic length an eigenvalue estimate based on work by Flanagan and Belytschko [15]. That characteristic length provides a stable time step, but in many cases is far more conservative than the minimum distance between nodes. For a cubic element with side length equal to 1, and thus also surface area of each face and volume equal to 1, the minimum distance between nodes is 1. However, the eigenvalue estimate is $1/\sqrt{3}$, which is only 58% of the minimum distance. As the length of the element is increased in one direction while keeping surfaces in the lateral direction squares of area 1, the eigenvalue estimate asymptotes to $1/\sqrt{2}$ for very long elements. If the length is decreased, the eigenvalue estimate asymptotes to the minimum distance between nodes for very thin elements. In this case, the eigenvalue estimate is always more conservative than the minimum distance between nodes. However, consider an element whose cross section in one direction is not a square but a trapezoid with one side length much greater than the other. Assume the large side length is 1 and the other side length is arbitrarily small, $\varepsilon$. In this case, the minimum distance between nodes becomes $\varepsilon$, creating a very small and inefficient time step. However, the eigenvalue estimate is related to the length across the middle of the trapezoid, which for the conditions stated is $1/2$. Since both distances provide stable time steps, and one or the other can be much larger in various circumstances, the most efficient calculation is obtained by using the maximum of the two lengths, either the eigenvalue estimate or the minimum distance between nodes, to determine the time step.

By using the maximum of the lengths, the computed critical time step should be at the edge of instability, and the `TIME STEP SCALE FACTOR` command line should be used to provide a margin of safety. In this case the scale factor for the time step should not be greater than 0.9, and in some cases it may have to be reduced further. Thus, although the maximum of the lengths provides a time step that is closer to the critical value and provides better accuracy and efficiency, you may need to specify a smaller-than-expected scale factor for stability. For this reason, the choice of which approach to use is left to the user and is determined by the command line

```
ELEMENT NUMERICAL FORMULATION = old | new (old) .
```

If the input parameter is `old`, only the eigenvalue estimate is used; while `new` means that the maximum of the two lengths is used. The default is `old` so that users will have to specifically choose the `new` approach and be aware of the scale factor for the time step.

The `ELEMENT NUMERICAL FORMULATION` command line is applicable to both the energy-dependent and purely mechanical material models. If this command line is applied to blocks using energy-dependent materials, only the determination of the characteristic length is affected. If this command line is applied to an element block with a purely mechanical model and the `old` option is used, the Pronto3D-based artificial viscosity, time step, and eigenvalue estimate will be used in the element calculations. If, however, the `new` option is used, the artificial viscosity and time step will be computed from equations associated with the energy-dependent models. You *should* consult Reference 7 for further details about the critical time-step calculations and the use of this command line.

### 2.3.11 Deactivate All Elements in an Element Block

```
DEACTIVATE = <string>code_name
```

This command line will be implemented in the future.

## 2.4 Presto Region and Procedure

The command blocks and command lines described in Section 2.3 are physics independent, and they reside in the domain scope. There are also command blocks and command lines that are Presto specific. These will appear in the region scope. The region scope must rest inside the procedure scope (see Section 1.1 for more information about scope). To create the scope for the Presto region and Presto procedure, use the following commands:

```
BEGIN PRESTO PROCEDURE <string>presto_procedure_name
  #
  # TIME CONTROL command block
  #
  BEGIN PRESTO REGION <string>presto_region_name
    #
    # command blocks and command lines that appear in the
    # region scope
    #
  END [PRESTO REGION <string>presto_region_name]
END [PRESTO PROCEDURE <string>presto_procedure_name]
```

Currently, only the `TIME CONTROL` command block appears within the `PRESTO PROCEDURE` command block and outside of the `PRESTO REGION` command block. These three command blocks are discussed below. See Section 2.5 through Section 2.15 for descriptions of the command lines and command blocks that will appear in the `PRESTO REGION` command block.

### 2.4.1 Presto Procedure

The entire time-stepping process, from the initial time to the termination time, is controlled within the procedure scope defined by the `PRESTO PROCEDURE` command block. The command block begins with

```
BEGIN PRESTO PROCEDURE <string>presto_procedure_name
```

and is terminated with

```
END [PRESTO PROCEDURE <string>presto_procedure_name] .
```

The string `presto_procedure_name` is the name for the Presto procedure.

### 2.4.2 Time Control

This section describes the input in Presto for time control of an analysis. The time control sets the begin and end times for the analysis and also controls incrementing the time step.

#### 2.4.2.1 Command Blocks for Time Control and Time Stepping

```
BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK <string>time_block_name
    START TIME = <real>start_time_value
    BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
```

```
        #
        # Time control parameters specific to PRESTO
        # are set in this command block.
        #
      END [PARAMETERS FOR PRESTO REGION <string>region_name]
    END [TIME STEPPING BLOCK <string>time_block_name]
    TERMINATION TIME = <real>termination_time ,
END [TIME CONTROL]
```

Presto time control resides in a `TIME CONTROL` command block. The command block begins with

```
BEGIN TIME CONTROL
```

and terminates with

```
END [TIME CONTROL] .
```

Within the `TIME CONTROL` command block, a number of `TIME STEPPING BLOCK` command blocks can be defined. Each `TIME STEPPING BLOCK` command block contains the time at which the time stepping starts and a number of parameters that set time-related values for the analysis. Each `TIME STEPPING BLOCK` command block terminates at the start time of the following command block. The start times for the `TIME STEPPING BLOCK` command blocks must be in increasing order. Otherwise, an error will be generated by Presto.

In the above input lines, the values are as follows:

- The string `time_block_name` is a name for the `TIME STEPPING BLOCK` command block. This name must be unique to the other command blocks of this type.
- The real value `start_time_value` is the start time for this `TIME STEPPING BLOCK` command block. Values set by the block apply from the start time for this block until the next start time or the termination time.
- The string `region_name` is the name of the Presto region affected by the parameters (see Section 2.4).

The final termination time for the analysis is given by the command line

```
TERMINATION TIME = <real>termination_time ,
```

where `termination_time` is the time at which the analysis will be stopped.

The `TERMINATION TIME` command line occurs after the last `TIME STEPPING BLOCK` command block is defined. Note that it is permissible to have `TIME STEPPING BLOCK` command blocks with start times after the termination time; in this case, those command blocks that have start times after the termination time are not executed. Only one `TERMINATION TIME` command line can appear. If more than one of these command lines appears, Presto gives an error.

Nested inside the TIME STEPPING BLOCK command block is a PARAMETERS FOR PRESTO REGION command block containing parameters that control the time stepping:

```
BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
  INITIAL TIME STEP = <real>initial_time_step_value
  TIME STEP SCALE FACTOR = <real>time_step_scale_factor
  TIME STEP INCREASE FACTOR =<real>time_step_increase_factor
  STEP INTERVAL = <integer>nsteps
END [PARAMETERS FOR PRESTO REGION <string>region_name] .
```

These parameters are specific to a Presto analysis.

The command block begins with

```
BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
```

and is terminated with

```
END [PARAMETERS FOR PRESTO REGION <string>region_name] .
```

As noted previously, the string region_name is the name of the Presto region affected by the parameters. The command lines nested inside the PARAMETERS FOR PRESTO REGION command block are described next.

#### 2.4.2.2 Initial Time Step

```
INITIAL TIME STEP = <real>initial_time_step_value
```

By default, Presto computes a critical time step for the analysis and uses this as the initial time step. To directly specify a different initial time step, use the INITIAL TIME STEP command line, where initial_time_step_value is the size of the initial time step. This command is only valid if it is in the first TIME STEPPING BLOCK command block in the problem.

The value for the initial time step will overwrite the calculated critical time step. If the user specifies an initial time step larger than the critical time step, the problem can become unstable.

#### 2.4.2.3 Time Step Scale Factor

```
TIME STEP SCALE FACTOR = <real>time_step_scale_factor(1.0)
```

During the element computations, Presto computes a minimum time step required for stability of the computation (the critical time step). Using the TIME STEP SCALE FACTOR command line, you can provide a scale factor to modify the critical time step. Note that a value greater than 1.0 for time_step_scale_factor will cause the time step to be greater than the computed critical time step, and thus the problem will likely go unstable. By default, the scale factor is 1.0.

#### 2.4.2.4 Time Step Increase Factor

```
TIME STEP INCREASE FACTOR =
```

```
<real>time_step_increase_factor(1.1)
```

During an analysis, the computed critical time step may change as elements deform, are killed, etc. By using the `TIME STEP INCREASE FACTOR` command line, you can limit the amount that the time step can increase between two adjacent time steps. The value `time_step_increase_factor` is a factor that multiplies the previous time step. The current time step can be no larger than the product of the previous time step and the scale factor.

If the computed time step is greater than the critical time step, Presto uses the computed limit instead. Note that an increase factor less than 1.0 will cause the time step to continuously decrease. The default value for this factor is 1.1, i.e., a time step cannot be more than 1.1 times the previous step.

#### 2.4.2.5 Step Interval

```
STEP INTERVAL = <integer>nsteps(100)
```

Presto can output data about the current time step, the current internal and external energy, and the kinetic energy throughout an analysis. The `STEP INTERVAL` command line controls the frequency of this output, where `nsteps` is the number of time steps between output. The default value for `nsteps` is 100.

The output at any given step (read from left to right) is

- step number,
- time,
- time increment,
- kinetic energy,
- internal energy,
- external energy (work done on boundary),
- error in energy balance,
- cpu time, and
- wall clock time.

The time is at the current time, step $n$, and the time increment is the previous time step increment from step $n - 1$ to step $n$.

The error in the energy balance is computed from the relation

```
energy balance error = (kinetic energy + internal energy
  - external energy) / external energy * 100 .
```

The above expression gives a percent error for the energy balance.

#### 2.4.2.6 Example

The following is a simple example of a `TIME CONTROL` command block:

```
BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK p1
    START TIME = 0.0
    BEGIN PARAMETERS FOR PRESTO REGION presto_region
      INITIAL TIME STEP = 1.0e-6
      STEP INTERVAL = 50
    END
  END
  BEGIN TIME STEPPING BLOCK p2
    START TIME = 0.5e-3
    BEGIN PARAMETERS FOR PRESTO REGION presto_region
      TIME STEP SCALE FACTOR = 0.9
      TIME STEP INCREASE FACTOR = 1.5
      STEP INTERVAL = 10
    END
  END
  TERMINATION TIME = 1.0e-3
END
```

### 2.4.3 Presto Region

Individual time steps are controlled within the region scope. The region scope is defined by a `PRESTO REGION` command block that begins with

```
BEGIN PRESTO REGION <string>presto_region_name
```

and is terminated with

```
END [PRESTO REGION <string>presto_region_name] .
```

The string `presto_region_name` is the name for the Presto region.

## 2.5 Use Finite Element Model

The model specification occurs within the region scope. To specify the model (finite element mesh), use the command line

```
USE FINITE ELEMENT MODEL <string>model_name .
```

The string `model_name` must match a name used in a `FINITE ELEMENT MODEL` command block described in Section 2.3. If one of these command blocks uses the name `penetrator` in the command-block line and this is the model we wish to use in the region, then we would enter the command line as

```
USE FINITE ELEMENT MODEL penetrator .
```

## 2.6 Kinematic Boundary Conditions

The various kinematic boundary conditions available in Presto are described in this section. The kinematic boundary conditions are nested inside the region scope.

### 2.6.1 Fixed Displacement Components

```
BEGIN FIXED DISPLACEMENT
  SURFACE = <string list>surface_names |
     NODE SET = <string list>nodelist_names
  COMPONENTS = <string>X/Y/Z
END [FIXED DISPLACEMENT]
```

The `FIXED DISPLACEMENT` command block fixes displacement components (X, Y, Z, or some combination thereof) for a set of nodes for all time. The nodes can be specified with either a `SURFACE` or `NODE SET` command line, but not both. Multiple surfaces or node sets can be listed within a single `SURFACE` or `NODE SET` command line.

The displacement components that are to be fixed are specified with the `COMPONENTS` command line.

### 2.6.2 Prescribed Displacement

```
BEGIN PRESCRIBED DISPLACEMENT
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED DISPLACEMENT]
```

The `PRESCRIBED DISPLACEMENT` command block prescribes a time history displacement (using a function) for a given set of nodes. The nodes can be specified with either a `SURFACE` or `NODE SET` command line, but not both. Multiple surfaces or node sets can be listed within a single `SURFACE` or `NODE SET` command line.

The time history displacement can be specified either along a component direction (X, Y, or Z) or in some arbitrary direction, but not both. The `COMPONENT` command line is used to specify that the displacement vector lies along one of the component directions, and the `DIRECTION` command line is used to specify that the displacement vector lies along an arbitrary direction. The direction option uses a `defined_direction` that has been defined in the domain scope.

The time history function is specified with the `FUNCTION` command line. This references a `function_name` (defined in the domain scope) that specifies the magnitude of the displacement vector as a function of time.

The displacement values in the function can be scaled by using the `SCALE FACTOR` command line. If the magnitude of the displacement in the time history function is given

as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the displacement from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

This boundary condition specifies the displacement only in the prescribed direction. It does not influence the displacement normal to the prescribed direction.

### 2.6.3 Prescribed Velocity

```
BEGIN PRESCRIBED VELOCITY
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z |
    CYLINDRICAL AXIS = <string>defined_axis |
    RADIAL AXIS = <string>defined_axis
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED VELOCITY]
```

The `PRESCRIBED VELOCITY` command block prescribes a time history velocity (using a function) for a given set of nodes. The nodes can be specified with either a `SURFACE` or `NODE SET` command line, but not both. Multiple surfaces or node sets can be listed within a single `SURFACE` or `NODE SET` command line.

The time history displacement can be specified either along a component direction (X, Y, or Z), in some arbitrary direction, along a radial direction (defined in reference to some axis), or along a cylindrical direction (defined in reference to some axis). Only one of these options is allowed. The `COMPONENT` command line is used to specify that the velocity vector lies along one of the component directions, and the `DIRECTION` command line is used to specify that the velocity vector lies along an arbitrary direction. The direction option uses a `defined_direction` that has been defined in the domain scope. The `CYLINDRICAL AXIS` command line requires an `axis` definition that appears in the domain scope. For this option, a radial line will be drawn from a node to the cylindrical axis. The velocity vector will lie along a path that is tangent to a circle that lies in a plane normal to the cylindrical axis and has a radius defined by the magnitude of the radial line from the node to the cylindrical axis. The `RADIAL AXIS` command line requires an `axis` definition that appears in the domain scope. For this option, a radial line is drawn from a node to the radial axis. The velocity vector lies along this radial line from the node to the radial axis.

The time history function is specified with the `FUNCTION` command line. This references a `function_name` (defined in the domain scope) that specifies the magnitude of the velocity as a function of time.

The velocity values in the function can be scaled by using the `SCALE FACTOR` command line. If the value of velocity in the time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the velocity from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

This boundary condition specifies the velocity only in the prescribed direction. It does not influence the velocity normal to the prescribed direction.

### 2.6.4 Prescribed Acceleration

```
BEGIN PRESCRIBED ACCELERATION
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED ACCELERATION]
```

The `PRESCRIBED ACCELERATION` command block prescribes a time history acceleration (using a function) for a given set of nodes. The nodes can be specified with either a `SURFACE` or `NODE SET` command line, but not both. Multiple surfaces or node sets can be listed within a single `SURFACE` or `NODE SET` command line.

The time history acceleration can be specified either along a component direction (X, Y, or Z) or in some arbitrary direction, but not both. The `COMPONENT` command line is used to specify that the acceleration vector lies along one of the component directions, and the `DIRECTION` command line is used to specify that the acceleration vector lies along an arbitrary direction. The direction option uses a `defined_direction` that has been defined in the domain scope.

The time history function is specified with the `FUNCTION` command line. This references a `function_name` (defined in the domain scope) that specifies the magnitude of the acceleration vector as a function of time.

The acceleration values in the function can be scaled by using the `SCALE FACTOR` command line. If the magnitude of the acceleration in the time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the acceleration from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

This boundary condition specifies the acceleration only in the prescribed direction. It does not influence the acceleration normal to the prescribed direction.

### 2.6.5 Fixed Rotation

```
BEGIN FIXED ROTATION
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  COMPONENTS = <string>X/Y/Z
END [FIXED ROTATION]
```

The `FIXED ROTATION` command block fixes rotation about direction components (X, Y, Z, or some combination thereof) for a set of nodes for all time. The nodes can be specified with either a `SURFACE` or `NODE SET` command line, but not both. Multiple surfaces or node sets can be listed within a single `SURFACE` or `NODE SET` command line.

The rotation components that are to be fixed are specified with the COMPONENTS command line. The rotation components for the nodes in the surface or node set are fixed for all time.

### 2.6.6 Prescribed Rotation

```
BEGIN PRESCRIBED ROTATION
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED ROTATION]
```

The PRESCRIBED ROTATION command block prescribes a time history for rotation about an axis for a given set of nodes. The nodes can be specified with either a SURFACE or NODE SET command line, but not both. Multiple surfaces or node sets can be listed within a single SURFACE or NODE SET command line.

The time history rotation can be specified either about a component direction (X, Y, or Z) or about some arbitrary direction, but not both. The COMPONENT command line is used to specify that the rotation vector is about one of the component directions, and the DIRECTION command line is used to specify that the rotation vector is about an arbitrary direction. The direction option uses a defined_direction that has been defined in the domain scope.

The time history function is specified with the FUNCTION command line. This references a function_name (defined in the domain scope) that specifies the magnitude of the rotation vector as a function of time.

The magnitude of the rotation in the function can be scaled by using the SCALE FACTOR command line. If the magnitude of the rotation in the time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the rotation from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The magnitude of the rotation, as specified by the product of the function and the scale factor, has units of radians per second.

## 2.7 Initial Conditions

An initial velocity can be specified for all nodes associated with a list of element blocks. This initial velocity condition is imposed with an `INITIAL VELOCITY` command block, which has the general form

```
BEGIN INITIAL VELOCITY
  BLOCK = <string list>block_name1 block_name2 ...
  # velocity specification command lines
END [INITIAL VELOCITY]
```

The `INITIAL VELOCITY` command block begins with

```
BEGIN INITIAL VELOCITY
```

and is terminated with

```
END INITIAL VELOCITY .
```

The element blocks are specified with the `BLOCK` command line. One or more element block names can be specified on the `BLOCK` command line. The velocity specification applies to all of the element blocks.

There are two different methods for specifying the initial velocity. These two methods are described in the following sections.

### 2.7.1 Initial Velocity Direction

```
BEGIN INITIAL VELOCITY
  BLOCK = <string list>block_names
  DIRECTION = <string>defined_direction
  MAGNITUDE = <real>magnitude_of_velocity
END [INITIAL VELOCITY]
```

The initial velocity can be specified in a given direction. The direction of the velocity vector is given by the `DIRECTION` command line with a defined direction that appears in the domain scope. The magnitude of the initial velocity is given by the `MAGNITUDE` command line with the real value `magnitude_of_velocity`.

If the direction option is selected for this command block, you must use both the `DIRECTION` command line and the `MAGNITUDE` command line. Neither of the command lines associated with the angular velocity option (`CYLINDRICAL AXIS` command line and `ANGULAR VELOCITY` command line) can appear in conjunction with the direction option.

### 2.7.2 Initial Angular Velocity

```
BEGIN INITIAL VELOCITY
  BLOCK = <string list>block_names
  CYLINDRICAL AXIS = <string>defined__axis
  ANGULAR VELOCITY = <real>angular_velocity
END [INITIAL VELOCITY]
```

The initial velocity can be specified as an initial angular velocity about some axis. The axis about which the body is initially rotating is given by the `CYLINDRICAL AXIS` command line with a defined axis that appears in the domain scope. The magnitude of the angular velocity about this axis is given by the `ANGULAR VELOCITY` command line with the real value `angular_velocity`.

For the angular velocity option, the angular velocity has units of radians per unit time. Typically, the value for the angular velocity will be radians per second.

If the angular velocity option is selected for this command block, you must use both the `CYLINDRICAL AXIS` command line and the `ANGULAR VELOCITY` command line. Neither of the command lines associated with the direction option (`DIRECTION` command line and `MAGNITUDE` command line) can appear in conjunction with the angular velocity option.

## 2.8 Force Boundary Conditions

There are a variety of force boundary conditions that are available in Presto. This section describes these boundary conditions.

### 2.8.1 Pressure

```
BEGIN PRESSURE
  SURFACE = <string list>surface_names
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESSURE]
```

The PRESSURE command block applies a pressure over a list of surfaces specified by the SURFACE command line. The pressure is in the opposite direction to the outward normals of the faces that define the surfaces. Currently, the PRESSURE command block can be used for surfaces that have faces derived from eight-node hexahedrons, four-node tetrahedrons, eight-node tetrahedrons, four-node membranes, and four-node shells. Only uniform pressure loads are handled at present. The magnitude of the pressure as a function of time is given by function_name on the FUNCTION command line. The function corresponding to function_name is defined in the domain scope. The pressure value is applied to all of the faces in the given surface.

The value of the pressure in the function can be scaled by using the SCALE FACTOR command line. If the magnitude of the pressure in the time history function is given as 100.0 from time 1.0 to time 2.0 and the scale factor is 2.5, then the magnitude of the pressure from time 1.0 to 2.0 is 250.0. The default value for the scale factor is 1.0.

### 2.8.2 Prescribed Force

```
BEGIN PRESCRIBED FORCE
  SURFACE = <string list>surface_names |
     NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED FORCE]
```

The PRESCRIBED FORCE command block prescribes a time history for forces on a given set of nodes. The nodes can be specified with either a SURFACE or NODE SET command line, but not both. Multiple surfaces or node sets can be listed within a single SURFACE or NODE SET command line.

The time-history nodal forces can be specified either along a component direction (X, Y, or Z) or in some arbitrary direction, but not both. The COMPONENT command line is used to specify that the force vector lies along one of the component directions, and the DIRECTION command line is used to specify that the force vector lies along an arbitrary

direction. The direction option uses a `defined_direction` that has been defined in the domain scope.

The time history function is specified with the `FUNCTION` command line. This references a `function_name` (defined in the domain scope) that specifies the magnitude of the force vector as a function of time.

The force values in the function can be scaled by using the `SCALE FACTOR` command line. If the magnitude of the force in the time history function is given as 100.0 from time 1.0 to time 2.0 and the scale factor is 2.5, then the magnitude of the force from time 1.0 to 2.0 is 250.0. The default value for the scale factor is 1.0.

### 2.8.3 Prescribed Moment

```
BEGIN PRESCRIBED MOMENT
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED MOMENT]
```

The `PRESCRIBED MOMENT` command block prescribes a time history for a moment about an axis for a given set of nodes. The nodes can be specified with either a `SURFACE` or `NODE SET` command line, but not both. Multiple surfaces or node sets can be listed within a single `SURFACE` or `NODE SET` command line.

The time history moment can be specified either about a component direction (X, Y, or Z) or about some arbitrary direction, but not both. The `COMPONENT` command line is used to specify that the moment vector is about one of the component directions, and the `DIRECTION` command line is used to specify that the moment vector is about an arbitrary direction. The direction option uses a `defined_direction` that has been defined in the domain scope.

The time history function is specified with the `FUNCTION` command line. This references a `function_name` (defined in the domain scope) that specifies the magnitude of the moment vector as a function of time.

The magnitude of the moment in the function can be scaled by using the `SCALE FACTOR` command line. If the magnitude of the rotation in the time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the rotation from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

## 2.9 Specialized Boundary Conditions

There are a number of specialized boundary conditions implemented in Presto. Some of them enforce kinematic conditions, and some result in the application of loads.

### 2.9.1 Gravity

```
BEGIN GRAVITY
  DIRECTION = <string>defined_direction
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
  GRAVITATIONAL CONSTANT = <real>g_constant
END [GRAVITY]
```

The `GRAVITY` command block is used to specify a gravity load that is applied to all element blocks in the finite element model. The direction of the gravity load is given by a defined direction on the `DIRECTION` command line. A gravitational constant is specified by the `GRAVITATIONAL CONSTANT` command line. (For example, the gravitational constant in units of inches and seconds would be 386.4 inches per second squared.) The strength of the gravitational field as a function of time can be varied by using the `FUNCTION` command line. This command line references a `function_name` defined in the domain scope. The dependent variables in the function can be scaled by the real value in the `SCALE FACTOR` command line. At any given time, the strength of the gravitational field is a product of the gravitational constant, the value of the function at that time, and the scale factor.

### 2.9.2 Cavity Expansion

```
BEGIN CAVITY EXPANSION
  SURFACE = <string>surface_name
  EXPANSION RADIUS = <string>spherical|
    cylindrical(spherical)
  TARGET NORMAL = <string>X|Y|Z
  FREE SURFACE = <real>upper_surface <real>lower_surface
  LAYER SURFACE = <real>upper_layer <real>lower_layer
  FREE SURFACE EFFECT COEFFICIENTS =
    <real>free_surf_coeff_top <real>free_surf_coeff_bottom
  PRESSURE COEFFICIENTS =
    <real>coeff1 <real>coeff2 <real>coeff3
  NODE SETS TO DEFINE BODY AXIS =
    <string>nodelist_id1 <string>nodelist_id2
  TIP RADIUS = <real>tip_radius
END [CAVITY EXPANSION]
```

The `CAVITY EXPANSION` command block is used to apply a cavity expansion boundary condition to a surface on a body. This boundary condition is used for earth penetration studies where some type of projectile strikes a target. For a more detailed explanation of the numerical implementation of the cavity expansion boundary condition and the parameters for this boundary condition, consult Reference 16. The cavity expansion boundary condition is a complex boundary condition with several options, and the detailed

explanation of the implementation of this boundary condition in Reference 16 is required reading to fully understand the input parameters for this boundary condition.

The boundary condition is applied to a surface (`surface_name`) specified by the `SURFACE` command line. This boundary condition generates a pressure at a node based on the velocity and surface geometry at the node. Since cavity expansion is essentially a pressure boundary condition, cavity expansion must be specified for a surface.

There are two types of cavity expansion—cylindrical expansion and spherical expansion. You can select either the spherical or cylindrical option by using the `EXPANSION RADIUS` command line; the default is `spherical`. Consult References 16 and 17 for more information about these two types of cavity expansion.

The normal to the target is specified by the `TARGET NORMAL` command line. The normal will be in the positive *x*-direction (`TARGET NORMAL = X`), the positive *y*-direction (`TARGET NORMAL = Y`), or the positive *z*-direction (`TARGET NORMAL = Z`).

The cavity expansion implementation accounts for surface effects. The free surfaces of the target are specified by the `FREE SURFACE` command line. The first value on this command line (`upper_surface`) defines the upper point on the target, and the second value (`lower_surface`) defines the lower point on the target. A layer in the target is specified with the `LAYER SURFACE` command line. The first value on this command line (`upper_layer`) defines the upper surface of the layer, and the second value on this command line (`lower_layer`) defines the lower surface of the layer. For further information about the layer specification, consult Reference 16.

While the surface depths for the free surface effects are set on the `FREE SURFACE` command line, the coefficients for these effects are provided in the `FREE SURFACE EFFECT COEFFICIENTS` command line. The first coefficient, `free_surf_coeff_top`, is for free surface effects at the top free surface; the second coefficient, `free_surf_coeff_bottom`, is for free surface effects at the bottom surface. If a surface effect coefficient is not zero, then the cavity expansion boundary condition will account for surface effects. If a surface effect coefficient is zero, then the boundary condition will NOT account for surface effects. For a detailed discussion of the surface effect calculations, consult Reference 16.

The value of the pressure at a node is derived from an equation that is quadratic based on some scalar value derived from the velocity vector at the node. The three coefficients for the quadratic equation are given by the `PRESSURE COEFFICIENTS` command line. The three coefficients (`coeff1`, `coeff2`, and `coeff3`) define the impact properties of the target. Consult Reference 17 for further information concerning these coefficients.

For the numerical implementation of cavity expansion, it is necessary to define two points on the axis (usually an axis of revolution) of the penetrator. These points are defined by the `NODE SETS TO DEFINE BODY AXIS` command line. The first point should be a node toward the forward tip of the penetrator (`nodelist_id1`); the second point should be a

node toward the aft end of the penetrator (`nodelist_id2`). Consult Reference 16 for more information about how the body axis is defined.

It is necessary to compute either a spherical or cylindrical radius for nodes on the surface where the cavity expansion boundary condition is applied. This is done automatically for most nodes. The calculations for these radii break down if the node is close to or at the axis of revolution of the body. For nodes where the radii calculations break down, a user-defined radius can be specified with the `TIP RADIUS` command line. For more information, consult Reference 16.

### 2.9.3 Periodic

```
BEGIN PERIODIC
  NODE SETS = <string>nodelist_id1 <string>nodelist_id2 |
    SURFACES = <string>surface_id1 <string>surface_id2
  SEARCH TOLERANCE = <real>search_tolerance(1.0e-4)
  PRESCRIBED QUANTITY = <string>DISPLACEMENT|VELOCITY|FORCE
  COMPONENT  = <string>X|Y|Z  |
    RADIAL AXIS = <string>defined_radial_axis
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
  THETA = <real>theta_value>
END [PERIODIC]
```

The `PERIODIC` command block is used to define boundary conditions for periodic structures. With the periodic boundary condition, it is possible to define only one segment of an object that is repeated on a periodic basis in order to model the entire object. Figure 2.1 shows an example of a structure that can be modeled using the periodic boundary condition. The basic shape, segment *n*, is repeated throughout the structure. Only this segment of the structure is actually modeled.

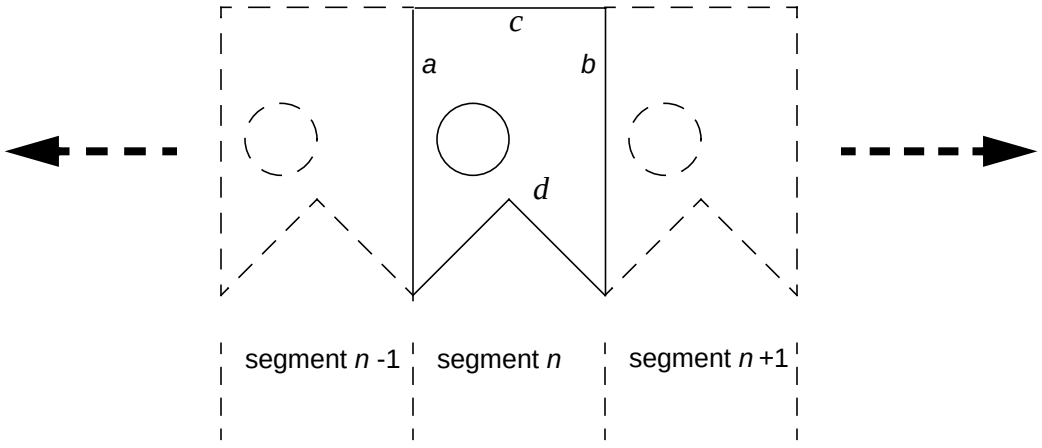



**Figure 2.1.** Periodic structure.

The boundary condition that is applied to the boundary *c* of segment *n* also applies to the corresponding boundaries of segments 1 through $n - 1$ and segments $n + 1$ through *m*, where $m > n + 1$. The same is also true for the boundary condition that applies to the boundary *d* of segment *n*. If, for example, a pressure boundary condition is applied to boundary *d* of segment *n*, then the same pressure boundary condition also applies to the corresponding boundary for all other segments. The boundaries *a* and *b* must have self-similar node sets modeling these boundaries. For example, if boundary *a* has ten nodes, these ten nodes must exactly overlay ten nodes on boundary *b* if the nodes on boundary *a* are simply moved by a translation from boundary *a* to boundary *b*.

Currently, only a limited set of options is available for the periodic boundary condition in Presto. Not all of the above options listed on the command lines are functional. Furthermore, the `PERIODIC` command block can only be used for serial analyses. This command block has not yet been implemented for parallel analyses.

The two self-similar node sets for the boundary condition are specified by the `NODE SETS` command line. It must be possible to replicate the coordinates for the nodes in one set (`nodelist_id1`) by a simple translation of the corresponding nodes in the other set (`nodelist_id2`). A geometric check is made to determine if the node sets are self-similar. The geometric check is set to a certain tolerance. This tolerance can be reset from a default value of $1 \times 10^{-4}$ by using the command line `SEARCH TOLERANCE`. If a node on boundary *a* lies within a radius defined by the search tolerance from a node on boundary *b*, the two nodes are considered self-similar.

Two self-similar node sets can also be specified by the `SURFACES` command line. A node set will be generated from a surface specification. All nodes associated with the specified surface will constitute the node set. It must be possible to replicate the coordinates for the nodes in one set (derived from the nodes associated with `surface_id1`) by a simple translation of the corresponding nodes in the other set (derived from the nodes associated with `surface_id2`). A geometric check is made to determine if the node sets are self-similar. This check for self-similarity is the same as that for the case of the node set specification.

You may use either a `NODE SETS` command line or a `SURFACES` command line, but not both. This command block is valid only for two node sets that are self-similar.

The periodic boundary condition forces the motion of two self-similar nodes to be the same. The motion of any two self-similar nodes must be the same to reflect the periodic nature of the structure. It is possible to prescribe the behavior for certain quantities on the self-similar boundaries *a* and *b*. The prescribed quantity that will exhibit periodic behavior on boundaries *a* and *b* is specified with the `PRESCRIBED QUANTITY` command line. At present, only the `DISPLACEMENT` option is allowed. This option forces the displacement component selected by the `COMPONENT` command line (either `X`, `Y`, or `Z`) to exhibit the periodic behavior. The prescribed behavior is defined by the `FUNCTION` command line. The `FUNCTION` command line references a `function_name` described in the domain

scope. The function is scaled by the scale_factor given on the the SCALE FACTOR command line.

Currently, the VELOCITY and FORCE options for the PRESCRIBED QUANTITY command line are not implemented. The RADIAL AXIS option is not implemented for the COMPONENT command line. The THETA command line will be used in conjunction with the RADIAL AXIS option, so it is not currently functional.

### 2.9.4 Silent Boundary

```
BEGIN SILENT BOUNDARY
  SURFACE = <string>surface_name
END [SILENT BOUNDARY]
```

The boundary condition defined by the SILENT BOUNDARY command block is also referred to as a nonreflecting surface boundary condition. A wave striking this surface is not reflected. This boundary condition is implemented with the techniques described in Reference 18. The method described in this reference is excellent at transmitting the low- and medium-frequency content through the boundary. While the method does reflect some of the high-frequency content, the amount of energy reflected is usually minimal. On the whole, the silent boundary condition implemented in Presto is highly effective.

To use the SILENT BOUNDARY command block, you only have to specify the surface that is nonreflective (surface_name) in the SURFACE command line.

### 2.9.5 Spot-Weld

```
BEGIN SPOT WELD
  NODE SET = <string>nodelist_id
  SURFACE = <surface>surface_id
  NORMAL DISPLACEMENT FUNCTION =
    <string>function_nor_disp
  NORMAL DISPLACEMENT SCALE FACTOR =
    <real>scale_nor_disp[1.0]
  TANGENTIAL DISPLACEMENT FUNCTION =
    <string>function_tang_disp
  TANGENTIAL DISPLACEMENT SCALE FACTOR =
    <real>scale_tang_disp[1.0]
  FAILURE ENVELOPE EXPONENT = <real>exponent
  FAILURE DECAY CYCLES = <integer>number_decay_cycles
END [SPOT WELD]
```

The spot-weld option lets the user model an “attachment” between a node on one surface and a face on another surface. This option models a weld or a small screw or bolt with a force-displacement curve like that shown in Figure 2.2. The displacement shown in the figure is the distance that the node moves from the nearest point on the face as measured in the original configuration. The force shown in the figure is the force at the attachment as a function of the distance between the two attachment points. (The force-displacement curve assumes the two attachment points are originally at the same location and the initial

distance is zero.) Two force-displacement curves are required for the spot-weld model. One curve models normal behavior, and the other curve models tangential behavior.

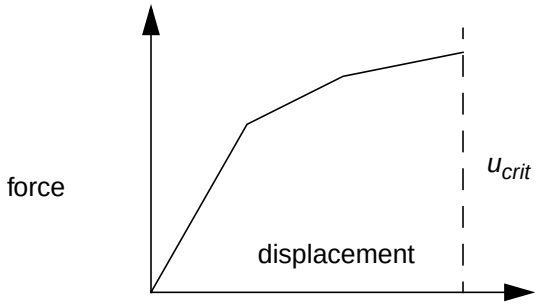



**Figure 2.2.** Force-displacement curve for spot-weld.

The attachment in Presto is defined between a node on one surface and the closest point on an element face on the other surface. Since a face is used to define one of the attachment points, it is possible to compute a normal vector and a tangent vector associated with the face. This allows us to resolve the displacement (distance) and force into normal and tangential components. With normal and tangential vectors associated with the attachment, the attachment can be characterized for the case of pure tension and pure shear. By specifying another parameter, a failure envelope exponent, we can control the interaction between the cases of pure tension and pure shear. For the spot-weld model implemented in Presto, the attachment remains intact as long as

$$(u_n / u_{n_{crit}})^p + (u_t / u_{t_{crit}})^p < 1.0. \tag{4}$$

In Equation (4), the distance from the node to the original attachment point on the face as measured normal to the face is $u_n$, which is defined as the normal distance. The maximum value given for $u_n$ in the normal force-displacement curve is $u_{n_{crit}}$. The distance from the node to the original attachment point on the face as measured along a tangent to the face is $u_t$, which is defined as the tangential distance. The maximum value given for $u_t$ in the tangential force-displacement curve is $u_{t_{crit}}$. In Figure 2.2, the maximum value for the displacement is $u_{crit}$.

To use the spot-weld option in Presto, a `SPOT WELD` command block begins with the input line

```
BEGIN SPOT WELD
```

and is terminated with the input line

```
END [SPOT WELD] .
```

Within the command block, it is necessary to specify the node on one surface with the `NODE SET` command line. Only one node (`nodelist_id`) can be in the node set

specified by the NODE SET command line. An element face (surface_id) on an opposing surface is specified with the SURFACE command line. Only one element face can be in the surface specified by the SURFACE command line. When calculating the closest point on the opposing surface to the node, this closest point should lie within the element face specified by the SURFACE command line.

The normal force-displacement curve is specified by a function named by the value function_nor_disp in the NORMAL DISPLACEMENT FUNCTION command line. This function can be scaled by the real value scale_nor_disp in the NORMAL DISPLACEMENT SCALE FACTOR command line; the default for this factor is 1.0. The tangential force-displacement curve is specified by a function named by the string function_tang_disp in the TANGENTIAL DISPLACEMENT FUNCTION command line. This function can be scaled by the real value scale_tang_disp given in the TANGENTIAL DISPLACEMENT SCALE FACTOR command line; the default for this factor is 1.0.

The failure envelope exponent, $p$ in Equation (4), is specified by the real value exponent in the FAILURE ENVELOPE EXPONENT command line.

For an explicit, transient dynamics code like Presto, it is better to remove the force for the spot-weld over several load steps rather than over a single load step once the failure criterion is exceeded. The FAILURE DECAY CYCLES command line controls the number of load steps over which the final force is removed. To remove the final force at a spot-weld over five load increments, the integer specified by number_decay_cycles would be set to 5. Once the force at the spot-weld is reduced to zero, it remains zero for all subsequent time.

# 2.10 Constraints

There are currently two different constraints available in Presto. These constraints maintain algebraic relations over time between a pair of nodes.

## 2.10.1 Constant Distance Constraint

```
BEGIN CONSTRAINT CONSTANT DISTANCE
  NODE SETS = <string list>nodelist_id1 nodelist_id2
  SEARCH TOLERANCE = <real>search_tol(0.0001)
END [CONSTRAINT CONSTANT DISTANCE]
```

The `CONSTRAINT CONSTANT DISTANCE` command block defines a constraint condition that preserves the distance between two nodes. If the original distance between two nodes at time 0 is *D*, the distance between the two nodes will remain *D* for all time.

Nodes are paired with the `NODE SETS` command line by specifying two separate node sets that are self-similar, i.e., one node set (`nodelist_id1`) will overlay the other node set (`nodelist_id2`) if either set is moved by a simple translation. The constraint condition pairs nodes that lie within a distance (`search_tol`) specified by the `SEARCH TOLERANCE` command line. The default value for `search_tol` is 0.0001. The constraint condition will produce an error if the node sets are not self-similar. All paired nodes are subjected to the distance constraint.

This constraint condition only works for serial runs at present.

## 2.10.2 Hex Shell Constraint

```
BEGIN CONSTRAINT HEX SHELL <string>constraint_name
  HEX SHELL BLOCK = <string>hexshell_block_id
  EXCLUDE SURFACE = <string>surface_id
END [CONSTRAINT HEX SHELL <string>contraint_name]
```

The `CONSTRAINT HEX SHELL` command block defines a constraint condition that will preserve the distance between two nodes that define a through-the-thickness edge of a hexahedral shell element. For transient dynamics, it is necessary to introduce this constraint for the hexahedral shell elements. If the original distance between two nodes on a hexahedral shell that define a through-the-thickness edge at time 0 is *D*, the distance between the two nodes will remain *D* for all time.

If you specify a hexahedral shell block with the command line

```
HEX SHELL BLOCK = <string>hexshell_block_id ,
```

then all through-the-thickness edges will be found, and the constraint will be automatically applied. The algorithm to detect the through-the-thickness edges makes use of the fact that a block of hexahedral shell elements will be only one element thick. Ambiguous cases, however, can occur. A single element tab protruding from a block of elements, for example, generates an ambiguous case. You can eliminate ambiguities by

using the EXCLUDE SURFACE command line. Any edges in the defined surface (surface_id) are excluded from the constraint condition. Note that it is the user's responsibility to determine whether there are ambiguous situations in the mesh or other special cases where the constraint should be eliminated. The user must then set up the proper surface definitions in the mesh and use these definitions in the CONSTRAINT HEX SHELL command block.

The hexahedral shell is currently under development and has not been implemented in Presto.

## 2.11 Mass Property Calculations

It is possible to request initial mass properties for an element block or a group of element blocks. This is done by using the MASS PROPERTIES command block:

```
BEGIN MASS PROPERTIES
  BLOCK = <string>block_id1 <string>block_id2 ...
  STRUCTURE NAME = <string>structure_name
END [MASS PROPERTIES] .
```

If only one element block (block_id1) is specified on the BLOCK command line, only the mass properties for that block are calculated. If several element blocks are specified on the BLOCK command line, then that collection of blocks will be treated as one entity, and the mass properties for that single entity are calculated. If, for example, two element blocks are listed (block_id1 and block_id2), the total mass for the two element blocks will be reported as the total mass property.

The MASS PROPERTIES command block reports the total mass for the structure as defined by the BLOCK command line and the center of mass for the structure in the global coordinate system X, Y, Z. It also reports the moments of inertia (*Ixx*, *Iyy*, *Izz*, *Ixy*, *Iyz*, *Izx*) about the center of mass of the structure in terms of the global coordinate system. The output for the mass properties will be identified by the structure_name specified for the structure on the STRUCTURE NAME command line.

This command block appears in the region scope.

# 2.12 Contact

This section describes the input syntax for defining interactions of contact surfaces in an analysis. For more information on contact and its computational details, consult Reference 2.

Within the contact scope, there are command lines and command blocks that define the specifics for the interaction of surfaces via the contact algorithm. Some of the command lines and command blocks within the contact scope set up default parameters that affect all contact calculations. Some of the command blocks in the contact scope affect only the interaction between a pair of surfaces.

There are three approaches that you can use to define a contact problem: (1) define a set of parameters that will apply to all contact interactions; (2) define a set of parameters that will apply to all contact interactions, and then override the defaults for a limited set of interactions; and (3) define all interactions separately such that each interaction can be defined by its own set of parameters. Which approach you choose will depend on the level of detail required to define all of the interactions.

You can have contact enforcement for all surfaces that are defined as contact surfaces or for only a limited set of surfaces defined as contact surfaces. If you use the `DEFAULTS` command block described in Section 2.12.8, all of the surfaces listed as contact surfaces will be treated as contact surfaces. If you omit the `DEFAULTS` command block, only those surfaces appearing in the `INTERACTION` command blocks described in Section 2.12.9 will be treated as contact surfaces.

The general pattern of syntax for describing contact is as follows:

- Identify all surfaces that need to be considered for contact. This is done with command lines within the contact scope.
- Specify any special contact options such as initial overlap removal, angle for multiple interactions, and number of enforcement iterations. This is done with command lines within the contact scope.
- Describe friction models used in the contacts for this analysis. Current types of friction models include frictionless contact, contact with constant coulomb friction, and tied contact. A friction model is described with a command block.
- Specify any rigid surface used for contact. Rigid surfaces are described with a command block.
- Set default values. Default values that apply to all of the surfaces interactions are specified in a command block. A normal tolerance and tangential tolerance are required input. The overlap removal tolerances and the default friction model can be set in the `DEFAULTS` command block. Automatic kinematic partitioning can be set as a global option in the `DEFAULTS` command block.
- Specify values for interactions between specific contact surfaces. This is done within a command block. Values specified in this command block can override

defaults for the normal and tangential tolerances, the friction model, and the kinematic partition factors.

### 2.12.1 Contact Definition Block

All commands for contact occur within a `CONTACT DEFINITION` command block. A summary of these commands follows.

```
BEGIN CONTACT DEFINITION <string>block_name
  CONTACT SURFACE <string>name
    CONTAINS <string list>surfaces
  CONTACT ALL BLOCKS
  REMOVE INITIAL OVERLAP
  MULTIPLE INTERACTIONS WITH ANGLE
    <real>angle(60.0)
  NUMBER OF ITERATIONS
    <integer>number_enforce_iter(5)
  BEGIN FRICTIONLESS MODEL <string>name
  END [FRICTIONLESS MODEL <string>name]
  BEGIN CONSTANT FRICTION MODEL <string>name
    FRICTION COEFFICIENT = <real>coeff
  END [CONTACT FRICTION MODEL <string>name]
  BEGIN TIED MODEL <string>name
  END [TIED MODEL <string>name]
  BEGIN ANALYTIC PLANE [<string>name]
    NORMAL = <string>defined_direction
    POINT = <string>defined_point
  END [ANALYTIC PLANE <string>name]
  BEGIN ANALYTIC CYLINDER [<string>name]
    CENTER = <string>defined_point
    AXIAL DIRECTION = <string>defined_axis
    RADIUS = <real>cylinder_radius
    LENGTH = <real>cylinder_length
    CONTACT NORMAL = <string>OUTSIDE|INSIDE
  END [ANALYTIC CYLINDER <string>name]
  BEGIN ANALYTIC SPHERE [<string>name]
    CENTER = <string>defined_point
    RADIUS = <real>sphere_radius
  END [ANALYTIC SPHERE <string>name]
  BEGIN DEFAULTS [<string>name]
    NORMAL TOLERANCE = <real>norm_tol
    TANGENTIAL TOLERANCE = <real>tang_tol
    OVERLAP NORMAL TOLERANCE = <real>norm_tol
    OVERLAP TANGENTIAL TOLERANCE = <real>tang_tol
    FRICTION MODEL = <string>name
    AUTOMATIC KINEMATIC PARTITION
  END [DEFAULTS <string>name]
  BEGIN INTERACTION [<string>name]
    MASTER = <string>surface
    SLAVE = <string>surface
    SURFACES = <string>surface1 <string>surface2
    KINEMATIC PARTITION = <real>kin_part
    NORMAL TOLERANCE = <real> norm_tol
    TANGENTIAL TOLERANCE = <real>tang_tol
    OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
    OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
    FRICTION MODEL = <string>name
```

```
    AUTOMATIC KINEMATIC PARTITION
  END [INTERACTION <string>name]
END [CONTACT DEFINITION <string>block_name]
```

The command block begins with the input line

```
BEGIN CONTACT DEFINITION <string>name
```

and is terminated with the input line

```
END [CONTACT DEFINITION <string>name] ,
```

where `name` is a name for this contact definition. The name should be unique among all the definitions of contact for an analysis. All other contact commands are encapsulated within this command block, as shown in the summary of the block presented previously. These other contact commands are described in Section 2.12.2 through Section 2.12.9. Section 2.12.10 gives an example of a valid contact definition.

A typical analysis will have only one `CONTACT DEFINITION` command block. However, more than one contact definition can be used. Each `CONTACT DEFINITION` command block creates its own contact entity. Therefore, fewer of these command blocks provide more efficient contact processing.

## 2.12.2 Descriptions of Contact Surfaces

All surfaces that have the potential for interaction through contact must be identified as contact surfaces. Presto input includes two ways to identify what surfaces are to be included for contact computations. The `CONTACT DEFINITION` command block MUST include some type of surface definition, either by use of the `CONTACT SURFACE` (Section 2.12.2.1) command line or the `CONTACT ALL BLOCKS` (Section 2.12.2.2) command line.

Any given face may NOT be included in more than one contact surface. See comments in the following two sections.

### 2.12.2.1 Contact Surface

```
CONTACT SURFACE <string>name CONTAINS <string list>surfaces
```

The first approach for identifying contact surfaces is offered by the `CONTACT SURFACE` command line. This command line identifies a set of surfaces (side sets) and element blocks that will be considered as a single contact surface; the string `name` is a name for this contact surface. This name should be unique to the other named contact surfaces within this `CONTACT DEFINITION` command block. The name on the `CONTACT SURFACE` command line (`name`) is used to refer to this surface in the interaction definitions described below. The list denoted by `surfaces` is a list of strings identifying the surfaces in the mesh file that are to be associated with this contact surface name. The surfaces can be side sets or element blocks or any combination of the two. Any specified element blocks are “skinned,” i.e., a surface is created from the exterior of the element block. If an element block is not contiguous to any adjacent element blocks (no shared

faces with other element blocks) and it is skinned, all of the exterior faces on the element block will appear in the list of contact faces. If an element block has shared faces with other element blocks and it is skinned, the shared faces will NOT appear in the list of faces defining the contact surface for the skinned element block.

The surfaces can contain a heterogeneous set of face types, and can contain any number of side sets and element blocks.

If a face appears in a side set and also appears in a set of faces generated by the skinning of an element block, this will produce an error. As indicated earlier, any given face may not appear in more than one contact surface.

#### 2.12.2.2 Contact All Blocks

```
CONTACT ALL BLOCKS
```

The second approach for identifying contact surfaces is offered by a single command line: `CONTACT ALL BLOCKS`. Often an analyst may wish to consider contact between the external surfaces of all the element blocks in the mesh. This command line causes all element blocks to be "skinned," i.e., a surface is created from the exterior of each element block. The skinned surfaces are then given contact surface names identical to the name of the element block. For instance, if a mesh contained the element blocks block_1, block_10, and block_11, then `CONTACT ALL BLOCKS` would create three contact surfaces from these blocks with the names block_1, block_10, and block_11, respectively. This approach is useful for large models in which the individual specification of contact surfaces would be unwieldy.

If the `CONTACT ALL BLOCKS` command line is used, contact surfaces cannot be defined by the above `CONTACT SURFACE` command line. The use of the `CONTACT ALL BLOCKS` command line would include all exposed faces on all element blocks in the contact set. A `CONTACT SURFACE` command line would include at least one exposed face on an element block. Specifying the same face in two different contact surfaces is not allowed.

### 2.12.3 Remove Initial Overlap

```
REMOVE INITIAL OVERLAP
```

Meshes supplied for finite element analyses frequently have some level of initial mesh overlap, where finite element nodes rest inside the volume of elements not connected to the node. This can cause problems with contact; overlaps may cause initial forces that are nonphysical and potentially destabilizing. To remove these initial overlaps, Presto provides the option of modifying the initial mesh to remove overlaps in surfaces defined for contact. The `REMOVE INITIAL OVERLAP` command line is used to conduct this operation.

This command line only removes overlaps that are detected along the surfaces defined for contact, not all surfaces in the mesh. If this command line is used, the normal and

tangential tolerances for the removal of overlap can be specified either in the DEFAULTS command block (Section 2.12.8) or in INTERACTION command blocks (Section 2.12.9). Overlap removal tolerances specified in INTERACTION command blocks will overwrite the default tolerances.

### 2.12.4 Angle for Multiple Interactions

```
MULTIPLE INTERACTIONS WITH ANGLE <real>angle(60.0)
```

Resolving contact interactions between a node penetrating near the edge between two adjoining faces can occur in one of two ways. If the angle between the two faces is small, then contact will identify one of the two faces to interact with, ignoring the other. However, in cases where the angle between the faces is large enough such that they form a discrete corner, returning a single interaction can yield poor results. In these cases, it is better to create two interactions—one with each face. However, the contact package can properly handle only a limited number of interactions per node (currently three), so it is in general not always feasible to have an interaction between a node and every face at a corner.

To provide some control over when a corner is "significant," i.e., when multiple interactions should be returned instead of just one interaction, Presto defines a critical angle for multiple interactions via the MULTIPLE INTERACTIONS WITH ANGLE command line, where angle is the angle over which an intersection between faces is considered sharp. If the angle between adjoining faces is greater than this critical angle, multiple interactions are created. By default, this critical angle is 60 degrees, which works well for most analyses. This value can be changed in the contact input if needed.

### 2.12.5 Iterative Enforcement

```
NUMBER OF ITERATIONS = <integer>num_iterations(5)
```

During the enforcement phase of the contact, an iterative process can be used to align two surfaces coming into contact. Rather than making one pass to compute contact forces for node push-back, several passes can be made so as to reduce overlap error for contact surfaces. The number of passes (iterations) is set by the value num_iterations in the NUMBER OF ITERATIONS command line. The default value for the number of iterations is 5. This command line affects only the enforcement phase of the contact. A single search phase is used for contact detection, but the enforcement phase can use an iterative process.

### 2.12.6 Friction Models

To describe the type of interactions that occur between contact surfaces, the Presto input for contact relies upon the definition of friction models. These models may then be applied to all or some subset of the potential surface interactions defined for contact. The currently available friction models are frictionless, constant coulomb friction, and tied contact. By default, interactions between contact surfaces that have not had friction models assigned are treated as frictionless. All friction models are command blocks, although some of the

models do not have any commands inside the command block. The commands for defining the available friction models are described next.

#### 2.12.6.1 Frictionless Model

```
BEGIN FRICTIONLESS MODEL <string>name
END [FRICTIONLESS MODEL <string>name]
```

The `FRICTIONLESS MODEL` command block defines frictionless contact between surfaces. No command lines are needed inside the command block. In the above input lines, `name` is a name assigned to this friction model.

#### 2.12.6.2 Constant Friction Model

```
BEGIN CONSTANT FRICTION MODEL <string>name
  FRICTION COEFFICIENT = <real>coeff
END [CONTACT FRICTION MODEL <string>name]
```

The `CONSTANT FRICTION MODEL` command block defines a constant coulomb friction coefficient between two surfaces as they slide past each other in contact. No resistance is provided to keep the surfaces together if they start to separate. In the above input lines, `name` is a name assigned to this friction model, and `coeff` is the constant coulomb friction coefficient.

#### 2.12.6.3 Tied Model

```
BEGIN TIED MODEL <string>name
END [TIED MODEL <string>name]
```

The `TIED MODEL` command block restricts nodes found in initial contact with faces to stay in the same relative location to the faces throughout the analysis. No command lines are needed inside the command block. In the above input lines, `name` is a name assigned to this friction model.

### 2.12.7 Analytic Contact Surfaces

Presto permits the definition of rigid analytic surfaces for use in contact. Contact evaluation between a deformable body and a rigid analytic surface is much faster than contact evaluation between two deformable bodies. Therefore, using a rigid analytic surface is more efficient than using a very stiff deformable body to try to approximate a rigid surface. The commands for defining the rigid analytic surfaces currently available in Presto—plane, cylinder, and sphere—are described next.

#### 2.12.7.1 Plane

```
BEGIN ANALYTIC PLANE [<string>name]
  NORMAL = <string>defined_direction
  POINT = <string>defined_point
END [ANALYTIC PLANE <string>name]
```

Analytic planes are not deformable, they cannot be moved, and two analytic planes will not interact with each other. The ANALYTIC PLANE command block for defining an analytic plane begins with the input line

```
BEGIN ANALYTIC PLANE [<string>name]
```

and is terminated with the input line

```
END [ANALYTIC PLANE <string>name] ,
```

where the string name is some user-selected name for this particular plane. This name, however, is not used internally by the code and is therefore optional. The string defined_direction in the NORMAL command line refers to a vector that has been defined with a DEFINE DIRECTION command line; this vector defines the outward normal to the plane. The string defined_point in the POINT command line refers to a point in a plane that has been defined with a DEFINE POINT command line. The deformable body should initially be on the side of the plane defined by the outward normal.

### 2.12.7.2 Cylinder

```
BEGIN ANALYTIC CYLINDER [<string>name]
  CENTER = <string>defined_point
  AXIAL DIRECTION = <string>defined_axis
  RADIUS = <real>cylinder_radius
  LENGTH = <real>cylinder_length
  CONTACT NORMAL = <string>OUTSIDE|INSIDE
END [ANALYTIC CYLINDER <string>name]
```

Analytic cylindrical surfaces are not deformable, they cannot be moved, and two analytic cylindrical surfaces will not interact with each other. The ANALYTIC CYLINDER command block for defining an analytic cylindrical surface begins with the command line

```
BEGIN ANALYTIC CYLINDER [<string>name]
```

and is terminated with the command line

```
END [ANALYTIC CYLINDER <string>name] ,
```

where the string name is some user-selected name for this particular cylindrical surface. This name, however, is not used internally by the code and is therefore optional. The cylindrical surface has a finite length; the cylindrical surface is not an infinitely long surface. To fully specify the location of the cylindrical surface, therefore, you must specify the center point of the cylindrical surface in addition to the axial direction of the cylinder. These quantities, center point and direction, are defined by the CENTER and AXIAL DIRECTION command lines, respectively. The string defined_point in the CENTER command line refers to a point that has been defined with a DEFINE POINT command line; the string defined_axis in the AXIAL DIRECTION command line refers to a vector that has been defined with a DEFINE DIRECTION command line. The radius of the cylinder is the real value cylinder_radius specified with the RADIUS command line, and the length of the cylinder is the real value cylinder_length specified by the LENGTH command line. The length of the cylinder (cylinder_length) extends a

distance of cylinder_length divided by 2 along the cylinder axis in both directions from the center point. If the rigid surface is the outside of the cylinder, you should specify

```
CONTACT NORMAL = OUTSIDE .
```

If the rigid surface is the inside of the cylinder, you should specify

```
CONTACT NORMAL = INSIDE .
```

#### 2.12.7.3 Sphere

```
BEGIN ANALYTIC SPHERE [<string>name]
  CENTER = <string>defined_point
  RADIUS = <real>sphere_radius
END [ANALYTIC SPHERE <string>name]
```

Analytic spherical surfaces are not deformable, they cannot be moved, and two analytic spherical surfaces will not interact with each other. The ANALYTIC SPHERE command block for defining an analytic spherical surface begins with the input line

```
BEGIN ANALYTIC SPHERE [<string>name]
```

and is terminated with the input line

```
END [ANALYTIC SPHERE <string>name] ,
```

where the string name is some user-selected name for this particular spherical surface. This name, however, is not used internally by the code and is therefore optional. The center point of the sphere is defined by the CENTER command line, which references a point defined_point specified by a DEFINE POINT command line. The radius of the sphere is the real value sphere_radius specified with the RADIUS command line.

### 2.12.8 Default Values for Interactions

```
BEGIN DEFAULTS [<string>name]
  NORMAL TOLERANCE = <real>norm_tol
  TANGENTIAL TOLERANCE = <real>tang_tol
  OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
  OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
  FRICTION MODEL = <string>name
  AUTOMATIC KINEMATIC PARTITION
END [DEFAULTS <string>name]
```

The DEFAULTS command block defines default values for the interactions between all contact surfaces, including interactions between a contact surface and itself (self-contact). If a DEFAULTS command block is provided, interactions between all the specified contact surfaces are defined. Without a DEFAULTS command block, only specifically defined interactions are defined. Only one DEFAULTS command block is permitted in a CONTACT DEFINITION command block. Some of the values for interactions have system defaults; values defined within a DEFAULTS command block override the system defaults.

The command block begins with the input line

```
BEGIN DEFAULTS [<string>name]
```

and ends with the input line

```
END [DEFAULTS <string>name] ,
```

where `name` is a name for the `DEFAULTS` command block. Note that this name is currently not used or required.

The valid commands within a `DEFAULTS` command block are described in Section 2.12.8.1 through Section 2.12.8.4. The values specified by these commands are applied by default to all interaction contact surfaces, unless overridden by a specific interaction definition. Normal and tangential tolerances, including the overlap normal and tangential tolerances, must always be specified by the user at some point. There are no default values for the normal and tangential tolerances since these are mesh-dependent values. All surfaces default to the frictionless contact model. The default for the kinematic partition factor for all surfaces is 0.5.

### 2.12.8.1 Normal and Tangential Tolerance

```
NORMAL TOLERANCE = <real>norm_tol
TANGENTIAL TOLERANCE = <real>tang_tol
```

The contact capabilities within Presto use a box defined around each face to locate nodes that may potentially contact the face. This box is defined by a tolerance normal to the face and another tolerance tangential to the face (see Figure 2.3). In the above command lines, `norm_tol` is the normal tolerance for the search box and `tang_tol` is the tangential tolerance for the search box.

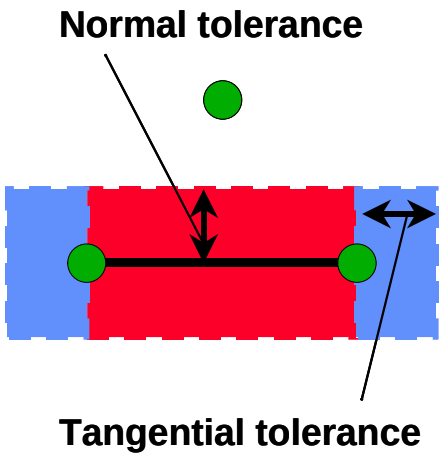



**Figure 2.3.** Illustration of normal and tangential tolerances.

Both of these tolerances are absolute distances in the same units as the analysis. The proper tolerances are problem dependent. There are no system defaults for these

tolerances; they must be specified either in a DEFAULTS command block or in an INTERACTION command block.

If a normal tolerance is specified in the DEFAULTS command block, the tolerance applies to all interactions. The default normal tolerance can be overwritten for a specific interaction by specifying a value for the normal tolerance for that interaction inside the BEGIN INTERACTION command block. The same is true for the tangential tolerance.

If no default normal tolerance is specified, then all surface interactions must have a normal tolerance specified. The same is true for the tangential tolerance.

#### 2.12.8.2 Normal and Tangential Overlap Tolerance

```
OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
```

The Presto contact input also permits the separate definition of normal and tangential search tolerances for the overlap removal option. Depending on the problem, the tolerances required to remove initial overlap may need to be different from those required for standard contact detection. In the OVERLAP NORMAL TOLERANCE command line, over_norm_tol is the normal tolerance for the search box for the removal of initial overlap. In the OVERLAP TANGENTIAL TOLERANCE command line, over_tang_tol is the tangential tolerance for the search box for the removal of initial overlap.

Both of these tolerances are absolute distances in the same units as the analysis. The proper tolerances for initial overlap are problem dependent. These tolerances must be chosen in such a way that they remove critical mesh overlap while not collapsing any small features in the mesh. Furthermore, the removal of the mesh overlap only occurs over a single element; if the mesh penetration is more than one element deep, removing overlap may invert elements.

If an overlap normal tolerance is specified in the DEFAULTS command block, the tolerance applies to all interactions. The default overlap normal tolerance can be overwritten for a specific interaction by specifying a value for the overlap normal tolerance for that interaction inside the BEGIN INTERACTION command block. The same is true for the overlap tangential tolerance.

If no default overlap normal tolerance is specified, then all surface interactions must have an overlap normal tolerance specified. The same is true for the overlap tangential tolerance.

#### 2.12.8.3 Friction Model

```
FRICTION MODEL = <string>name
```

The FRICTION MODEL command line permits the description of how surfaces interact with each other using a friction model defined in a CONTACT DEFINITION command

block (see Section 2.12.6). In the above command line, `name` is the name of the friction model to apply, as defined in a `CONTACT DEFINITION` command block.

As a system default, all interactions are defined as frictionless contact.

#### 2.12.8.4 Automatic Kinematic Partition

```
AUTOMATIC KINEMATIC PARTITION
```

If the `AUTOMATIC KINEMATIC PARTITION` command line is used, the contact package will automatically compute the kinematic partition factors for surfaces (Section 2.12.9.2). The kinematic partitions are computed based on nodal average density and wave speed. The partition factors are exact when the opposing surfaces have the same mesh resolution.

For the interaction of any two surfaces, the sum of the partition factors for the surfaces must be 1.0. This is automatically taken care of when the `AUTOMATIC KINEMATIC PARTITION` command line is used. The default value for kinematic partition factors for all surfaces is 0.5.

The `AUTOMATIC KINEMATIC PARTITION` command line can be used to set the kinematic partitions for all interactions or to set the kinematic partitions for specific interactions. Thus the command line can appear in two different scopes:

1. The command line can be used within the `DEFAULTS` command block. In this case, all contact surface interactions will use the automatic kinematic partitioning scheme by default. This will override the default case that assigns a kinematic partition factor of 0.5 to all surfaces. For particular interactions, it is possible to override the use of the automatic kinematic partition factors by specifying kinematic partition values (with the `KINEMATIC PARTITION` command line) within the `INTERACTION` command blocks for those interactions.

2. The command line can be used inside an `INTERACTION` command block. If the automatic partitioning command line appears inside an `INTERACTION` command block, the kinematic partition factors for that particular interaction will be calculated by the automatic kinematic partition scheme.

### 2.12.9 Values for Specific Interactions

```
BEGIN INTERACTION [<string>name]
  MASTER = <string>surface
  SLAVE = <string>surface
  SURFACES = <string>surface1 <string>surface2
  KINEMATIC PARTITION = <real>kin_part
  NORMAL TOLERANCE = <real> norm_tol
  TANGENTIAL TOLERANCE = <real>tang_tol
  OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
  OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
  FRICTION MODEL = <string>name
  AUTOMATIC KINEMATIC PARTITION
END [INTERACTION <string>name]
```

The Presto contact input also permits the setting of values for specific interactions using the INTERACTION command block. If a DEFAULTS command block is present within a CONTACT DEFINITION command block, the values provided by a an INTERACTION command block override the defined defaults. If a DEFAULTS command block is not present, only those interactions described by INTERACTION command blocks are searched for contact, and values without system defaults must be specified.

The INTERACTION command block begins with

```
BEGIN INTERACTION [<string>name]
```

and ends with

```
END [INTERACTION <string>name] ,
```

where name is a name for the interaction. Note that this name is currently not used or required.

All of the tolerance and friction model commands described in Section 2.12.8.1 through Section 2.12.8.3 are valid for this command block, as well as the commands described below in Section 2.12.9.1 and Section 2.12.9.2. The same is true for the AUTOMATIC KINEMATIC PARTITION command line described in Section 2.12.8.4.

### 2.12.9.1 Surface Identification

```
MASTER = <string>surface
SLAVE = <string>surface

SURFACES = <string>surface1 <string>surface2
```

There are two methods to identify the surfaces described by a specific interaction. To specify a one-way interaction, where the nodes of the “slave” surfaces are searched against the “master” surface, use the MASTER and SLAVE command lines, where surface is the name of a contact surface defined in the CONTACT DEFINITION command block (see Section 2.12.2). In this case, all other values specified (tolerances, friction model, kinematic partition) are applied only to nodes of the slave surface interacting with faces of the master surface.

Alternatively, both surfaces can be given in a single line with the SURFACES command line, where surface1 and surface2 are the names of the two contact surfaces to which the interaction refers. In this syntax, the values supplied for the interaction are defined for the two-way interaction (i.e., both the first surface as master and the second as slave, and vice versa).

Either of these syntaxes can be used for self-contact. For the MASTER and SLAVE command lines, both master and slave can be the same surface. Similarly, the two surfaces given in the SURFACES command line can be the same contact surface.

#### 2.12.9.2 Kinematic Partition

```
KINEMATIC PARTITION = <real>kin_part
```

The KINEMATIC PARTITION command line permits partitioning of the enforcement of contact between the two surfaces. This capability is important in cases where contact occurs between two materials of highly disparate stiffness. Physically, we would expect a material with a higher stiffness to have more of an effect in determining the position of the contact surface than a more compliant material. We would then assign a higher kinematic partition to the stiffer material.

Another case where the kinematic partition can become important is when meshes with dissimilar resolutions contact each other. If an interaction is defined with a fine mesh as the master surface and a coarse mesh as a slave surface, the contact algorithms may permit nodes on the master surface to penetrate the slave surface. By increasing the kinematic partition factor on the coarse mesh, the magnitude of this penetration can be reduced.

The real value `kin_part` in the command line is the kinematic partition factor. If the interaction is specified using the MASTER and SLAVE line commands, the kinematic partition is the factor for the slave nodes. Note that if both interactions are specified, i.e., two interactions are defined with opposite master/slave definitions, the kinematic partitions for the two interactions must add up to 1.0 so that the full contact force is applied. If the interaction is specified using the SURFACES command line, the kinematic partition factor refers to the second surface, and the first surface automatically receives a kinematic partition of `1.0 - kin_part`.

For self-contact, the kinematic partition factor can be set to 0.5 or 0.0. If the kinematic partition factor is set to 0.0, the self-contact is turned off.

### 2.12.10 Example

The following contact definition is valid and demonstrates the use of the above commands.

```
# contact definition for problem
BEGIN CONTACT DEFINITION contact_def

  # define contact surfaces
  CONTACT SURFACE surf_1 CONTAINS block_1
  CONTACT SURFACE surf_2 CONTAINS surface_2 block_2
  CONTACT SURFACE surf_3 CONTAINS block_3 block_4

  # set up removal of initial overlap
  REMOVE INITIAL OVERLAP

  # define friction models
  BEGIN FRICTIONLESS MODEL no_friction
  END
  BEGIN CONSTANT FRICTION MODEL some_friction
    FRICTION COEFFICIENT = 0.5
  END

  # define defaults for contact interactions
```

```
  BEGIN DEFAULTS
    NORMAL TOLERANCE = 0.1
    TANGENTIAL TOLERANCE = 0.05
    OVERLAP NORMAL TOLERANCE = 0.01
    OVERLAP TANGENTIAL TOLERANCE = 0.005
    FRICTION MODEL = some_friction
  END

  # define some specific contact interactions
  #   interactions between surf_2 and surf_3
  BEGIN INTERACTION
    SURFACES = surf_2 surf_3
    KINEMATIC PARTITION = 0.2
    FRICTION MODEL = no_friction
    NORMAL TOLERANCE = 0.2
  END

  #   self contact on surf_1
  BEGIN INTERACTION
    SURFACES = surf_1 surf_1
    OVERLAP NORMAL TOLERANCE = 0.2
    OVERLAP TANGENTIAL TOLERANCE = 0.1
  END
END
```

## 2.13 Results Output

```
BEGIN RESULTS OUTPUT <string>results_name
  DATABASE NAME = <string>results_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  NODE VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  NODAL VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  ELEMENT VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  GLOBAL VARIABLES  = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
END [RESULTS OUTPUT <string>results_name]
```

You can specify a results file, the results to be included in this file, and the frequency at which results are written by using a `RESULTS OUTPUT` command block. The command block appears inside the region scope.

More than one results file can be specified for an analysis. Thus for each results file, there will be one `RESULTS OUTPUT` command block. The command block begins with

```
BEGIN RESULTS OUTPUT <string>results_name
```

and is terminated with

```
END [RESULTS OUTPUT <string>results_name] ,
```

where `results_name` is a user-selected name for the command block. Nested within the `RESULTS OUTPUT` command block are a set of command lines, as shown in the block summary given above. The first two command lines listed (`DATABASE NAME` and `DATABASE TYPE`) give pertinent information about the results file. The command line

```
DATABASE NAME = <string>results_file_name
```

gives the name of the results file with the string `results_file_name`. If the results file is to appear in the current directory and is named `job.e`, this command line would appear as

```
DATABASE NAME = job.e .
```

If the results file is to be created in some other directory, the command line would have to show the path to that directory.

If the results file does not use the Exodus II format, you must specify the format for the results file using the command line

```
DATABASE TYPE = <string>database_type(exodusII) .
```

Currently, only the Exodus II database is supported by the SIERRA Framework. Other options will be added in the future.

The other command lines that appear in the `RESULTS OUTPUT` command block determine the type and frequency of information that is output. Descriptions of these command lines follow in Section 2.13.1 through Section 2.13.10.

### 2.13.1 Output Nodal Variables

```
NODE VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name
  <string>variable_name AS <string>dbase_variable_name ...]

NODAL VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name
  <string>variable_name AS <string>dbase_variable_name ...]
```

Any registered nodal variable in Presto can be selected for output in the results file by using a command line in one of the two forms shown above. The only difference between the two forms is the use of `NODE` or `NODAL`. The registered variables are listed in Appendix B.

It is possible to specify an alias for any of the registered nodal variables by using the `AS` specification. Suppose, for example, you wanted to output the external forces in Presto, which are registered as `force_external`, with the alias `f_ext`. You would then enter the command line

```
NODE VARIABLES = force_external AS f_ext .
```

The `NODE VARIABLES` command line can be used any number of times within a `RESULTS OUTPUT` command block. It is also possible to specify more than one nodal variable for output on a command line. If you also wanted to output the internal forces, which are registered as `force_internal`, with the alias `f_int`, you would enter the command line

```
NODE VARIABLES = force_external AS f_ext
  force_internal AS f_int .
```

The specification of an alias is always optional.

### 2.13.2 Output Element Variables

```
ELEMENT VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name
  <string>variable_name AS <string>dbase_variable_name ...]
```

Any registered element variable in Presto can be selected for output in the results file by using the `ELEMENT VARIABLES` command line. The registered variables are listed in Appendix B.

It is possible to specify an alias for any of the registered element variables by using the `AS` specification. Suppose, for example, you wanted to output the stress in Presto, which is registered as `rotated_stress`, with the alias `stress`. You would then enter the command line

```
ELEMENT VARIABLES = rotated_stress AS stress .
```

The `ELEMENT VARIABLES` command line can be used any number of times within a `RESULTS OUTPUT` command block. It is also possible to specify more than one element variable for output on a command line. If you also wanted to output the stretch, which is registered as `stretch`, with the alias `stretch`, you would enter the command line

```
ELEMENT VARIABLES = rotated_stress AS stress
  stretch AS stretch .
```

The specification of an alias is always optional.

### 2.13.3 Output Global Variables

```
GLOBAL VARIABLES  = <string>variable_name
  [AS <string>dbase_variable_name
  <string>variable_name AS <string>dbase_variable_name ...]
```

Any registered global variable in Presto can be selected for output in the results file by using the `GLOBAL VARIABLES` command line. The registered variables are listed in Appendix B. With the `AS` specification, you can specify the variable and select an alias for this variable in the results file. Suppose, for example, you wanted to output the time steps in Presto, which are identified as `timestep`, with the alias `tstep`. You would then enter the command line

```
GLOBAL VARIABLES = timestep AS tstep .
```

The `GLOBAL VARIABLES` command line can be used any number of times within a `RESULTS OUTPUT` command block. It is also possible to specify more than one global variable for output on a command line. If you also wanted to output the kinetic energy, which is registered as `KineticEnergy`, with the alias `ke`, you would enter the command line

```
GLOBAL VARIABLES = timestep as tstep
  KineticEnergy as ke .
```

The specification of an alias is always optional.

### 2.13.4 Set Begin Time for Results Output

```
START TIME = <real>output_start_time
```

Using the `START TIME` command line, you can write output to the results file beginning at time `output_start_time`. No results will be written before this time. If other commands set times for results (`AT TIME`, `ADDITIONAL TIMES`) that are less than `output_start_time`, those times will be ignored, and results will not be written at those times.

### 2.13.5 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that the output will be at exactly the times specified. To hit the output times exactly in an explicit, transient dynamics code, it is necessary to adjust the time step as the time approaches an output time. The integer value `steps` in the `TIMESTEP ADJUSTMENT INTERVAL` command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, results are output at times closest to the specified output times.

### 2.13.6 Output Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by `time_begin`, results will be output every time increment given by the real value `time_increment_dt`.

### 2.13.7 Additional Times for Output

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any times specified by the command line in Section 2.13.6, you can use the `ADDITIONAL TIMES` command line to specify an arbitrary number of additional output times.

### 2.13.8 Output Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by `step_begin`, results will be output every step increment given by the integer value `step_increment`.

### 2.13.9 Additional Steps for Output

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 2.13.8, you can use the `ADDITIONAL STEPS` command line to specify an arbitrary number of additional output steps.

### 2.13.10 Set End Time for Results Output

```
TERMINATION TIME = <real>termination_time_value
```

Results will not be written to the results file after time `termination_time_value`. If other commands set times for results (`AT TIME`, `ADDITIONAL TIMES`) that are greater than `termination_time_value`, those times will be ignored, and results will not be written at those times.

## 2.14 History Output

```
BEGIN HISTORY OUTPUT <string>history_name
  DATABASE NAME = <string>history_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  VARIABLE = <string>entity_type
    <string>internal_name
    AT <string>entity_type <integer>entity_id
    AS <string>history_variable_name
  VARIABLE = <string>entity_type_global
    <string>internal_name
    AS <string>history_variable_name
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
END [HISTORY OUTPUT <string>history_name]
```

A history file gives nodal results (displacement, force_external, etc.) for specific nodes and global results at specified times. You can specify a history file, the results to be included in this file, and the frequency at which results are written by using a `HISTORY OUTPUT` command block. The command block appears inside the region scope.

More than one history file can be specified for an analysis. For each history file, there will be one `HISTORY OUTPUT` command block. The command block for a history file description begins with

```
BEGIN HISTORY OUTPUT <string>history_name
```

and is terminated with

```
END [HISTORY OUTPUT <string>history_name] ,
```

where `history_name` is a user-selected name for the command block. Nested within the `HISTORY OUTPUT` command block are a set of command lines, as shown in the block summary given above. The first two command lines listed (`DATABASE NAME` and `DATABASE TYPE`) give pertinent information about the history file. The command line

```
DATABASE NAME = <string>history_file_name
```

gives the name of the history file with the string `history_file_name`. If the history file is to appear in the current directory and is named `job.e`, this command line would appear as

```
DATABASE NAME = job.e .
```

If the history file is to be created in some other directory, the command line would have to show the path to that directory.

If the history file does not use the Exodus II format, you must specify the format for the history file using the command line

```
DATABASE TYPE = <string>database_type(exodusII) .
```

Currently, only the Exodus II database is supported by the SIERRA Framework. Other options will be added in the future.

The other command lines that appear in the `HISTORY OUTPUT` command block determine the type and frequency of information that is output. Descriptions of these command lines follow in Section 2.14.1 through Section 2.14.8. Note that the command lines for controlling the frequency of history output (in Section 2.14.2 through Section 2.14.8) are the same as those for controlling the frequency of results output. These frequency-related command lines are repeated here for convenience.

## 2.14.1 Output Variables

The `VARIABLE` command line is used to select registered variables for output in the history file. Two forms of this command line are available, depending on the type of variable. The first form selects registered nodal and element variables. The second form selects global variables only.

### 2.14.1.1 Nodal and Element Output Variables

```
VARIABLE = <string>entity_type <string>internal_name
  AT <string>entity_type <integer>entity_id
  [AS <string>history_variable_name]
```

This form of the `VARIABLE` command line lets you select any registered nodal or element variable in Presto for output in the history file. The registered variables are listed in Appendix B. The variable is identified in the command line by setting `entity_type` to `NODE` (or `NODAL`) or `ELEMENT` and is selected with the string `internal_name`. In addition to an entity type, you must select a specific entity (node or element number) with the integer quantity `entity_id`. You can also specify an arbitrary name, `history_variable_name`, for the selected entity. For example, suppose you want to output the accelerations at node 88. The command line to obtain the accelerations at node 88 for the history file would be

```
VARIABLE = NODE ACCELERATION AT NODE 88 AS accel_88 ,
```

where `accel_88` is the arbitrary name that will be used for this history variable in the history file.

Note that either the key word `NODE` or `NODAL` can be used for nodal quantities. The specification of an alias is always optional.

### 2.14.1.2 Global Output Variables

```
VARIABLE = <string>entity_type_global
  <string>internal_name
```

```
[AS <string>history_variable_name]
```

This form of the VARIABLE command line lets you select any registered global variable in Presto for output in the history file. The registered variables are listed in Appendix B. The string entity_type_global in the command line can only be GLOBAL, and the registered variable is selected with the string internal_name. You can also specify an arbitrary name, history_variable_name, for the selected entity. For example, suppose you want to output the kinetic energy (KineticEnergy) as KE. The command line to obtain the kinetic energy in the history file would be

```
VARIABLE = GLOBAL KineticEnergy AS KE .
```

The specification of an alias is always optional.

### 2.14.2 Set Begin Time for History Output

```
START TIME = <real>output_start_time
```

Using the START TIME command line, you can write history variables to the history file beginning at time output_start_time. No history variables will be written before this time. If other commands set times for history output (AT TIME, ADDITIONAL TIMES) that are less than output_start_time, those times will be ignored, and history output will not be written at those times.

### 2.14.3 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that the output will be at exactly the times specified. To hit the output times exactly in an explicit, transient dynamics code, it is necessary to adjust the time step as the time approaches an output time. The integer value steps in the TIMESTEP ADJUSTMENT INTERVAL command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, history variables are output at times closest to the specified output times.

### 2.14.4 Output Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by time_begin, history variables will be output every time increment given by the real value time_increment_dt.

### 2.14.5 Additional Times for Output

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any times specified by the command line in Section 2.14.4, you can use the ADDITIONAL TIMES command line to specify an arbitrary number of additional output times.

### 2.14.6 Output Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by step_begin, history variables will be output every step increment given by the integer value step_increment.

### 2.14.7 Additional Steps for Output

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 2.14.6, you can use the ADDITIONAL STEPS command line to specify an arbitrary number of additional output steps.

### 2.14.8 Set End Time for History Output

```
TERMINATION TIME = <real>termination_time_value
```

History output will not be written to the history file after time termination_time_value. If other commands set times for history output (AT TIME, ADDITIONAL TIMES) that are greater than termination_time_value, those times will be ignored, and history output will not be written at those times.

## 2.15 Restart Data

```
BEGIN RESTART DATA <string>restart_name
  DATABASE NAME = <string>restart_file
  INPUT DATABASE NAME = <string>input_restart_file
  OUTPUT DATABASE NAME = <string>output_restart_file
  DATABASE TYPE = <string>database_type(exodusII)
  START TIME = <real>restart_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
END [RESTART DATA <string>restart_name]
```

You can specify restart files, either to be written to or read from, and the frequency at which restarts are written by using a `RESTART DATA` command block. The command block appears inside the region scope. To initiate a restart, the `RESTART TIME` command line (see Section 2.1.3.1) must also be used. This command line appears in the domain scope.

The `RESTART DATA` command block begins with the input line

```
BEGIN RESTART DATA <string>restart_name
```

and is terminated with

```
END [RESTART DATA <string>restart_name] ,
```

where `restart_name` is a user-selected name for the `RESTART DATA` command block.

Nested within the `RESTART DATA` command block are a set of command lines, as shown in the block summary given above. With the first command line listed, you can specify a database containing the input restart data, the output restart data, or both:

```
DATABASE NAME = <string>restart_file_name .
```

If the analysis is writing restart data, the data will be written to the file `restart_file_name`. The original restart file will be overwritten if it exists (after being read if applicable). If the file name begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name `restart_file_name`. If the restart file is to appear in the current directory and is named `job_restart.e`, this command line would appear as

```
DATABASE NAME = job_restart.e .
```

If the restart file is to be created in some other directory, the command line would have to show the path to that directory.

You can specify the database containing the input restart data by using the command line

```
INPUT DATABASE NAME = <string>restart_input_file .
```

If this analysis is being restarted, restart data will be read from this file. You can specify the database containing the output restart data by using the command line

```
OUTPUT DATABASE NAME = <string>restart_output_file .
```

If the analysis is writing restart data, the data will be written to this file. These latter two commands use the same file-naming convention as the command line `DATABASE NAME`.

If the restart file does not use the Exodus II format, you must specify the format for the results file using the `DATABASE TYPE` command line:

```
DATABASE TYPE = <string>database_type(exodusII) .
```

Currently, only the Exodus II database is supported by the SIERRA Framework. Other options will be added in the future.

The other command lines that appear in the `RESTART DATA` command block determine the frequency at which restarts are written. Descriptions of these command lines follow in Section 2.15.1 through Section 2.15.7. Note that the command lines for controlling the frequency of restart output are the same as those for controlling the frequency of results output and history output. These frequency-related command lines are repeated here for convenience.

## 2.15.1 Set Begin Time for Restart Writes

```
START TIME = <real>restart_start_time
```

Using the `START TIME` command line, you can write restarts to the restart output file beginning at time `restart_start_time`. No restarts will be written before this time. If other commands set times for restarts (`AT TIME`, `ADDITIONAL TIMES`) that are less than `restart_start_time`, those times will be ignored, and restarts will not be written at those times.

## 2.15.2 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that the restarts will be written at exactly the times specified. To hit the restart times exactly in an explicit transient dynamics code, it is necessary to adjust the time step as the time approaches a restart time. The integer value `steps` in the `TIMESTEP ADJUSTMENT INTERVAL` command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, then restarts are written at times closest to the specified restart times.

### 2.15.3 Restart Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by `time_begin`, restarts will be written every time increment given by the real value `time_increment_dt`.

### 2.15.4 Additional Times for Restart

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any restart times specified by the command line in Section 2.15.3, you can use the `ADDITIONAL TIMES` command line to specify an arbitrary number of additional restart times.

### 2.15.5 Restart Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by `step_begin`, restarts will be written every step increment given by the integer value `step_increment`.

### 2.15.6 Additional Steps for Restart

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 2.15.5, you can use the `ADDITIONAL STEPS` command line to specify an arbitrary number of additional restart steps.

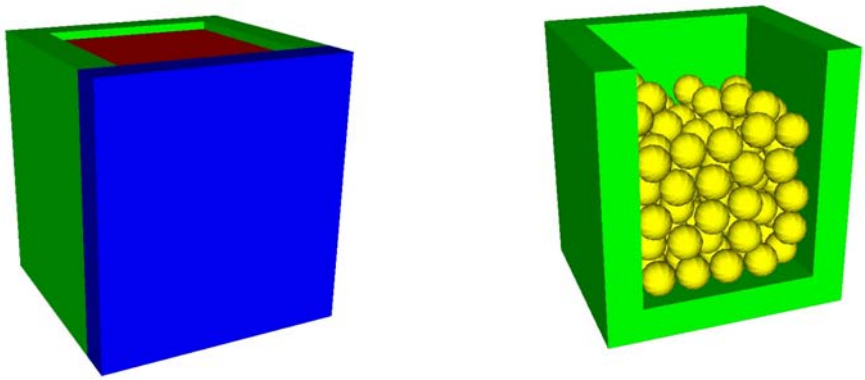
### 2.15.7 Set End Time for Restart Writes

```
TERMINATION TIME = <real>termination_time_value
```

Restarts will not be written to the restart output file after time `termination_time_value`. If other commands set times for restarts (`AT TIME`, `ADDITIONAL TIMES`) that are greater than `termination_time_value`, those times will be ignored, and restarts will not be written at those times.

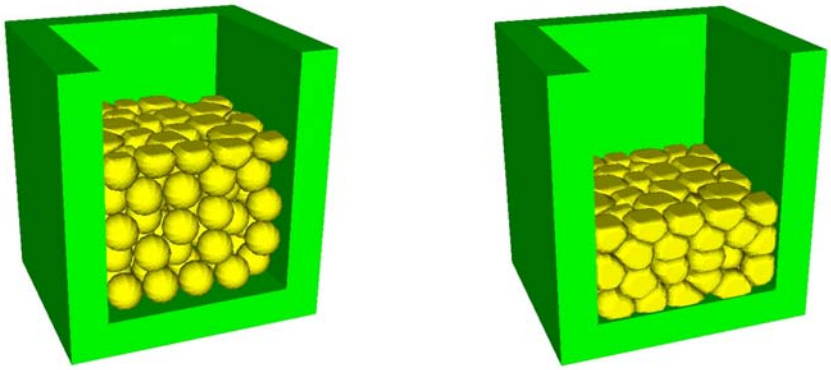
# 3 Example Problem

This section provides an example problem to illustrate the construction of an input file for an analysis. The example problem consists of 124 spheres made of lead enclosed in a steel box. The steel box has an open top into which a steel plate is placed (see Figure 3.1). A prescribed velocity is then applied on the steel plate, pushing it into the box and crushing the spheres contained within using frictionless contact. This problem is a severe test for the contact algorithms as the spheres crush into a solid block. See Figure 3.2 for results of this problem.



(a) (b)

**Figure 3.1.** Mesh for example problem: (a) Box (blue and green surfaces) with plate in top (red surface) and (b) Mesh with blue and red surfaces removed to show internal spheres (yellow).



t = 0.000333 second t = 0.0007 second

**Figure 3.2.** Results of Crush 124 Spheres test.

The input file is described below, with comments to explain every few lines. Following the description, the full input file is listed again. Most of the keywords in this example are all lowercase, which is different from the convention we have used to describe the command

lines in this document. However, all of the lowercase usage in the following example is an acceptable format in Presto.

The input file starts with a begin sierra statement, as is required for all input files:

```
begin sierra crush_124_spheres
```

We now need to define the functions used with this problem. The boundary conditions require a function for the initial velocity, as follows:

```
 begin definition for function constant_velocity
   type is piecewise linear
   ordinate is velocity
   abscissa is time
   begin values
     0.0 30.0
     1.0 30.0
   end values
 end definition for function constant_velocity
```

To define the boundary conditions, we need to define the direction for the initial velocity—this is in the *y*-direction. We could also choose to simply specify the Y component for the initial condition, but this input file uses directions.

```
 define direction y_axis with vector 0.0 1.0 0.0
```

Next we define the material models that will be used for this analysis. There are two materials in this problem: steel for the box, and lead for the spheres. Both materials use the elastic-plastic material model (denoted as `elastic_plastic`).

```
begin property specification for material steel
  density = 7871.966988

  begin parameters for model elastic_plastic
    youngs modulus = 1.999479615e+11
    poissons ratio = 0.33333
    yield stress = 275790291.7
    hardening modulus = 275790291.7
    beta = 1.0
  end parameters for model elastic_plastic

end property specification for material steel

begin property specification for material lead
  density = 11253.30062

  begin parameters for model elastic_plastic
    youngs modulus = 1.378951459e+10
    poissons ratio = 0.44
    yield stress = 13789514.59
    hardening modulus = 0.0
    beta = 1.0
  end parameters for model elastic_plastic

end property specification for material lead
```

Now, we define the finite element mesh. This includes specification of the file that contains the mesh, as well as a list of all the element blocks we will use from the mesh and the material associated with each block. The name of the file is crush_124_spheres.g. The specification of the database type is optional—ExodusII is the default. Currently, each element block must be defined individually. For this particular problem, all of the spheres are the same element block. Each sphere is a distinct geometry entity, but all spheres constitute one element block in the Exodus II database. Note that the three element blocks that make up the box and lid all reference the same material description. The material description is *not* repeated three times. The material description for steel appears once and is then referenced three times.

```
begin finite element model mesh1
  Database Name = crush_124_spheres.g
  Database Type = exodusII

  begin parameters for block block_1
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_1

  begin parameters for block block_2
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_2

  begin parameters for block block_3
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_3

  begin parameters for block block_4
    material linear_elastic_lead
    solid mechanics use model elastic_plastic
  end parameters for block block_4

end finite element model mesh1
```

At this point we have finished specifying physics-independent quantities. We now want to set up the Presto procedure and region, along with the `time control` command block. We start by defining the beginning of the procedure scope, the `time control` command block, and the beginning of the region scope. Only one `time stepping block` command block is needed for this analysis. The termination time is set to $7 \times 10^{-4}$.

```
begin presto procedure Apst_Procedure

begin time control
  begin time stepping block p1
    start time = 0.0
    begin parameters for presto region presto
      time step scale factor = 1.0
      time step increase factor = 2.0
      step interval = 25
    end parameters for presto region presto
  end time stepping block p1
```

```
  termination time = 7.0e-4
end time control

  begin presto region presto
```

Next we associate the finite element model we defined above (mesh1) with this presto region.

```
    use finite element model mesh1
```

Now we define the boundary conditions on the problem. We prescribe the velocity on the top surface of the box (nodelist_100) to crush the spheres, and we confine the bottom surface of the box (nodelist_200) not to move. Note that although we use node sets to define these boundary conditions, we could have used the corresponding side sets.

```
    begin prescribed velocity
      node set = nodelist_100
      direction = y_axis
      function = constant_velocity
      scale factor = -1.0
    end

    begin fixed displacement
      node set = nodelist_200
      components = Y
    end
```

Now we define the contact for this problem. For this problem, we want all four element blocks to be able to contact each other, with a normal tolerance of 0.0001 and a tangential tolerance of 0.0005. In this case, we simply define the same contact characteristics for all interactions. However, we could also specify tolerances and kinematic partition factors for individual interactions. Since no friction model is defined in the block below, the contact defaults to frictionless contact.

```
    begin contact definition
      contact all blocks
      begin defaults
        normal tolerance = 0.0001
        tangential tolerance = 0.00005
      end
    end
```

Now we define what variables we want in the results output file, as well as how often we want this file to be written. Here we request output files written every $7 \times 10^{-6}$ second of analysis time. This will result in results output at one hundred time steps (plus the zero time step) since the termination time is set to $7 \times 10^{-4}$ second. The output file will be called crush_124_spheres.e, and it will be an Exodus II file (the database type command is optional; it defaults to ExodusII). The variables we are requesting are the displacements and reactions at the nodes, the stresses for the elements, the time-step increment, and the kinetic energy.

```
    begin Results Output output_presto
```

```
          Database Name = crush_124_spheres.e
          Database Type = exodusII
          At Time 0.0, Increment = 7.0e-6
          nodal Variables = displacement as displ
          nodal Variables = reactions as react
          element Variables = stress
          global Variables = KineticEnergy as KE
          global Variables = timestep
        end
```

Now we end the presto region, presto procedure, and sierra blocks to complete the input file.

```
    end presto region presto
  end presto procedure Apst_Procedure
end sierra crush_124_spheres
```

Here is the resulting full input file for this problem:

```
begin sierra crush_124_spheres
   begin definition for function constant_velocity
     type is piecewise linear
     ordinate is velocity
     abscissa is time
     begin values
       0.0 30.0
       1.0 30.0
     end values
   end definition for function constant_velocity
   define direction y_axis with vector 0.0 1.0 0.0

  begin property specification for material steel
    density = 7871.966988

    begin parameters for model elastic_plastic
      youngs modulus = 1.999479615e+11
      poissons ratio = 0.33333
      yield stress = 275790291.7
      hardening modulus = 275790291.7
      beta = 1.0
    end parameters for model elastic_plastic

  end property specification for material steel

  begin property specification for material lead
    density = 11253.30062

    begin parameters for model elastic_plastic
      youngs modulus = 1.378951459e+10
      poissons ratio = 0.44
      yield stress = 13789514.59
      hardening modulus = 0.0
      beta = 1.0
    end parameters for model elastic_plastic

  end property specification for material lead

  begin finite element model mesh1
```

```
  Database Name = crush_124_spheres.g
  Database Type = exodusII

  begin parameters for block block_1
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_1

  begin parameters for block block_2
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_2

  begin parameters for block block_3
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_3

  begin parameters for block block_4
    material linear_elastic_lead
    solid mechanics use model elastic_plastic
  end parameters for block block_4

end finite element model mesh1

begin presto procedure Apst_Procedure

  begin time control
    begin time stepping block p1
      start time = 0.0
      begin parameters for presto region presto
        time step scale factor = 1.0
        time step increase factor = 2.0
        step interval = 25
      end parameters for presto region presto
    end time stepping block p1

    termination time = 7.0e-4
  end time control

  begin presto region Apst_Region

    use finite element model mesh1

    begin prescribed velocity
      node set = nodelist_100
      direction = y_axis
      function = constant_velocity
      scale factor = -1.0
    end prescribed velocity

    begin fixed displacement
      node set = nodelist_200
      components = Y
    end fixed displacement

    begin contact definition
      contact all blocks
```

```
        begin defaults
          normal tolerance = 0.0001
          tangential tolerance = 0.00005
        end
      end

      begin Results Output output_presto
        Database Name = crush_124_spheres.e
        Database Type = exodusII
        At Time 0.0, Increment = 7.0e-6
        nodal Variables = displacement as displ
        nodal Variables = reactions as react
        element Variables = stress
        global Variables = KineticEnergy as KE
        global Variables = timestep
      end results output output_presto

    end presto region presto
  end presto procedure Apst_Procedure
end sierra crush_124_spheres
```

# References


1. Edwards, H. C., and J. R. Stewart. “SIERRA: A Software Environment for Developing Complex Multi-Physics Applications.” In *First MIT Conference on Computational Fluid and Solid Mechanics*, edited by K. J. Bathe, 1147–1150. Amsterdam: Elsevier, 2001.

2. Brown, K. H., R. M. Summers, M. W. Glass, A. S. Gullerud, M. W. Heinstein, and R. E. Jones. *ACME: Algorithms for Contact in a Multiphysics Environment*, API Version, 1.0. Albuquerque, NM: Sandia National Laboratories, October 2001.

3. Schoof, L. A., and V. R. Yarberry. *EXODUS II: A Finite Element Data Model*, SAND92-2137. Albuquerque, NM: Sandia National Laboratories, September 1994.

4. Mitchell, J. A., A. S. Gullerud, W. M. Scherzinger, J. R. Koteras, and V. L. Porter. “ADAGIO: Non-Linear Quasi-Static Structural Response Using the SIERRA Framework.” In *First MIT Conference on Computational Fluid and Solid Mechanics*, edited by K. J. Bathe, 361–364. Amsterdam: Elsevier, 2001.

5. Stone, C. M. *SANTOS - A Two-Dimensional Finite Element Program for the Quasistatic, Large Deformation, Inelastic Response of Solids*, SAND90-0543. Albuquerque, NM: Sandia National Laboratories, 1996.

6. Whirley, R. G., B. E. Engelmann, and J. O. Halquist. *DYNA3D Users Manual*. Livermore, CA: Lawrence Livermore Laboratory, 1991.

7. Swegle, J. W. *SIERRA: PRESTO Theory Documentation: Energy Dependent Materials Version 1.0*. Albuquerque, NM: Sandia National Laboratories, October 2001.

8. Taylor, L. M., and D. P. Flanagan. *Pronto3D: A Three-Dimensional Transient Solid Dynamics Program*, SAND87-1912. Albuquerque, NM: Sandia National Laboratories, March 1989.

9. Krieg, R. D. *A Simple Constitutive Description for Cellular Concrete*, SAND SC-DR-72-0883. Albuquerque, NM: Sandia National Laboratories, 1978.

10. Swenson, D. V., and L. M. Taylor. “A Finite Element Model for the Analysis of Tailored Pulse Stimulation of Boreholes.” *International Journal for Numerical and Analytical Methods in Geomechanics* 7 (1983): 469–484.

11. Rashid, M. M. “Incremental Kinematics for Finite Element Applications.” *International Journal for Numerical Methods in Engineering* 36 (1993): 3937–3956.

12. Key, S. W., M. W. Heinstein, C. M. Stone, F. J. Mello, M. L. Blanford, and K. G. Budge. “A Suitable Low-Order, Tetrahedral Finite Element for Solids.” *International Journal for Numerical Methods in Engineering* 44 (1999) 1785–1805.

13. Key, S. W., and C. C. Hoff. “An Improved Constant Membrane and Bending Stress Shell Element for Explicit Transient Dynamics.” *Computer Methods in Applied Mechanics and Engineering* 124, no. 1–2 (1995): 33–47.

14. Laursen, T. A., S. W. Attaway, and R. I. Zadoks. *SEACAS Theory Manuals: Part III. Finite Element Analysis in Nonlinear Solid Mechanics*, SAND98-1760/3. Albuquerque, NM: Sandia National Laboratories, 1999.

15. Flanagan, D. P., and T. Belytschko. “A Uniform Strain Hexahedron and Quadrilateral with Orthogonal Hourglass Control.” *International Journal for Numerical Methods in Engineering* 17 (1981): 679–706.

16. Brown, K. H., J. R. Koteras, D. B. Longcope, and T. L. Warren. *CavityExpansion: A Library for Cavity Expansion Algorithms, Version 1.0*, in review. Albuquerque, NM: Sandia National Laboratories, 2003.

17. Warren, T. L., and M. R. Tabbara. *Spherical Cavity-Expansion Forcing Function in Pronto3D for Application to Penetration Problems*, SAND97-1174. Albuquerque, NM: Sandia National Laboratories, May 1997.

18. Lysmer, J., and R. L. Kuhlmeyer. “Finite Dynamic Model for Infinite Media.” *Journal of the Engineering Mechanics Division, Proceedings of the American Society of Civil Engineers* (August 1979): 859–877.

# Appendix A: Command Specification

This appendix gives all of the Presto commands in the proper scope.

```
# Domain specification

BEGIN SIERRA <string>name

  # Title

  TITLE = <string>title

  # Restart time

  RESTART TIME = <real>restart_time
  RESTART = AUTOMATIC

  # Function definition

  BEGIN DEFINITION FOR FUNCTION <string>function_name
    TYPE = <string>CONSTANT | PIECEWISE LINEAR
    ABSCISSA = <string>abscissa_label
    ORDINATE = <string>ordinate_label
    BEGIN VALUES
      <real>value_1  [<real>value_2
      <real>value_3   <real>value_4
      ...             <real>value_n]
    END [VALUES]
  END [DEFINITION FOR FUNCTION <string>function_name]

  # Definitions

  DEFINE POINT <string>point_name WITH COORDINATES
    <real>value_1 <real>value_2 <real>value_3

  DEFINE DIRECTION <string>direction_name WITH VECTOR
    <real>value_1 <real>value_2 <real>value_3

  DEFINE AXIS <string>axis_name WITH POINT
    <string>point_name DIRECTION <string>direction_name

  DEFINE AXIS <string>axis_name WITH POINT
    <string>point_1 POINT <string>point_2

  # Elastic material

  BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
    DENSITY = <real>density_value
    BEGIN PARAMETERS FOR MODEL ELASTIC
      YOUNGS MODULUS = <real>youngs_modulus
      POISSONS RATIO = <real>poissons_ratio
      SHEAR MODULUS = <real>shear_modulus
      BULK MODULUS = <real>bulk_modulus
      LAMBDA = <real>lambda
    END [PARAMETERS FOR MODEL ELASTIC]
  END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

  # Elastic-plastic material
```

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ELASTIC_PLASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING MODULUS = <real>hardening_modulus
    BETA = <real>beta_parameter(1.0)
  END [PARAMETERS FOR MODEL ELASTIC_PLASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Elastic-plastic power-law hardening

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL EP_POWER_HARD
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING CONSTANT = <real>hardening_constant
    HARDENING EXPONENT = <real>hardening_exponent
    LUDERS STRAIN = <real>luders_strain
  END [PARAMETERS FOR MODEL EP_POWER_HARD]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Soil and crushable foam

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL SOIL_FOAM
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    A0 = <real>const_coeff_yieldsurf
    A1 = <real>lin_coeff_yieldsurf
    A2 = <real>quad_coeff_yieldsurf
    PRESSURE CUTOFF = <real>pressure_cutoff
    PRESSURE FUNCTION = <string>function_press_volstrain
  END [PARAMETERS FOR MODEL SOIL_FOAM]
END [PROPERTY SPECIFICIATION FOR MATERIAL <string>mat_name

# Foam plasticity

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL FOAM_PLASTICITY
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
```

```
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    PHI = <real>phi
    SHEAR STRENGTH  = <real>shear_strength
    SHEAR HARDENING = <real>shear_hardening
    SHEAR EXPONENT  = <real>shear_exponent
    HYDRO STRENGTH  = <real>hydro_strength
    HYDRO HARDENING = <real>hydro_hardening
    HYDRO EXPONENT  = <real>hydro_exponent
    BETA = <real>beta
  END [PARAMETERS FOR MODEL FOAM_PLASTICITY]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Orthotropic crush

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    EX  = <real>modulus_x
    EY  = <real>modulus_y
    EZ  = <real>modulus_z
    GXY = <real>shear_modulus_xy
    GYZ = <real>shear_modulus_yz
    GZX = <real>shear_modulus_zx
    VMIN = <real>min_crush_volume
    CRUSH XX = <string>stress_strain_xx_function_name
    CRUSH YY = <string>stress_strain_yy_function_name
    CRUSH ZZ = <string>stress_strain_zz_function_name
    CRUSH XY =
      <string>shear_stress_strain_xy_function_name
    CRUSH YZ =
      <string>shear_stress_strain_yz_function_name
    CRUSH ZX =
      <string>shear_stress_strain_zx_function_name
  END [PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Orthotropic rate

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_RATE
    YOUNGS MODULUS = <real>youngs_modulus
    YIELD STRESS = <real>yield_stress
    MODULUS TTTT = <real>modulus_tttt
    MODULUS TTLL = <real>modulus_ttll
    MODULUS TTWW = <real>modulus_ttww
    MODULUS LLLL = <real>modulus_llll
    MODULUS LLWW = <real>modulus_llww
    MODULUS WWWW = <real>modulus_wwww
    MODULUS TLTL = <real>modulus_tltl
    MODULUS LWLW = <real>modulus_lwlw
```

```
    MODULUS WTWT = <real>modulus_wtwt
    TX = <real>tx
    TY = <real>ty
    TZ = <real>tz
    LX = <real>lx
    LY = <real>ly
    LZ = <real>lz
    MODULUS FUNCTION = <string>modulus_function_name
    RATE FUNCTION = <string>rate_function_name
    T FUNCTION = <string>t_function_name
    L FUNCTION = <string>l_function_name
    W FUNCTION = <string>w_function_name
    TL FUNCTION = <string>tl_function_name
    LW FUNCTION = <string>lw_function_name
    WT FUNCTION = <string>wt_function_name
  END [PARAMETERS FOR MODEL ORTHOTROPIC_RATE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

#Mie-Gruneisen Model

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN
    RHO_0 = <real>density
    C_0 = <real>sound_speed
    SHUG = <real>const_shock_velocity
    GAMMA_0 = <real>ambient_gruneisen_param
    POISSR = <real>poissons_ratio
    Y_0 = <real>yield_strength
    PMIN = <real>mean_stress(REAL_MAX)
  END [PARAMETERS FOR MODEL MIE_GRUNEISEN]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Mie-Gruneisen Power Series Model

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES
    RHO_0 = <real>density
    C_0 = <real>sound_speed
    K1 = <real>power_series_coeff1
    K2 = <real>power_series_coeff2
    K3 = <real>power_series_coeff3
    K4 = <real>power_series_coeff4
    K5 = <real>power_series_coeff5
    GAMMA_0 = <real>ambient_gruneisen_param
    POISSR = <real>poissons_ratio
    Y_0 = <real>yield strength
    PMIN = <real>mean_stress(REAL_MAX)
  END [PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# JWL (Jones-Wilkins-Lee) Model

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL JWL
    RHO_0 = <real>initial_density
    D = <real>detonation_velocity
    E_0 = <real>init_chem_energy
    A = <real>jwl_const_pressure1
```

```
    B = <real>jwl_const_pressure2
    R1 = <real>jwl_const_nondim1
    R2 = <real>jwl_const_nondim2
    OMEGA = <real>jwl_const_nondim3
    XDET = <real>x_detonation_point
    YDET = <real>y_detonation_point
    ZDET = <real>z_detonation_point
    TDET = <real>time_of_detonation
    B5 = <real>burn_width_const(2.5)
  END [PARAMETERS FOR MODEL JWL]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Ideal Gas Model

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL IDEAL_GAS
    RHO_0 = <real>initial_density
    C_0 = <real>initial_sound_speed
    GAMMA = <real>ratio_specific_heats
  END [PARAMETERS FOR MODEL IDEAL_GAS]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Define mesh

BEGIN FINITE ELEMENT MODEL <string>mesh_descriptor
  DATABASE NAME = <string>mesh_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  ALIAS <string>mesh_identifier AS <string>user_name
  BEGIN PARAMETERS FOR BLOCK <string>block_name
    MATERIAL <string>material_name
    SOLID MECHANICS USE MODEL <string>model_name
    ELEMENT STRAIN FORMULATION = <string>midpoint|
      strongly-objective(midpoint)
    LINEAR BULK VISCOSITY =
      <real>linear_bulk_viscosity_value(0.06)
    QUADRATIC BULK VISCOSITY =
      <real>quad_bulk_viscosity_value(1.20)
    HEX HOURGLASS STIFFNESS =
      <real>hour_glass_stiff_value(0.05)
    HEX HOURGLASS VISCOSITY =
      <real>hour_glass_visc_value(0.0)
    MEMBRANE SCALE THICKNESS =
      <real>mem_scale_thick_value
    SHELL SCALE THICKNESS = <real>shell_scale_thick_value
    SHELL INTEGRATION POINTS =
      <integer>number_integration_points(5)
    SHELL INTEGRATION SCHEME = <string>GAUSS|LOBATTO|
      TRAPEZOID(TRAPEZOID)
    TRUSS AREA = <real>truss_cross_sectional_area
    SOLID MECHANICS TEMPERATURE INITIAL =
      <real>init_temp_value
    SOLID MECHANICS TEMPERATURE FUNCTION =
      <string>defined_function
    DEPOSIT SPECIFIC INTERNAL ENERGY <real>edep
      OVER TIME tdep STARTING AT TIME tinit
    ELEMENT NUMERICAL FORMULATION = old | new (old)
    DEACTIVE = <string>code_name
  END [PARAMETERS FOR BLOCK <string>block_name]
```

```
END [FINITE ELEMENT MODEL <string>mesh_descriptor]

# Begin Procedure scope

BEGIN PRESTO PROCEDURE <string>procedure_name

  # Time block

  BEGIN TIME CONTROL
    BEGIN TIME STEPPING BLOCK <string>time_block_name
      START TIME = <real>start_time_value
      BEGIN PARAMETERS FOR PRESTO REGION
        <string>region_name
        INITIAL TIME STEP = <real>initial_time_step_value
        TIME STEP SCALE FACTOR =
        <real>time_step_scale_factor(1.0)
        TIME STEP INCREASE FACTOR =
        <real>time_step_increase_factor(1.1)
        STEP INTERVAL = <integer>nsteps(100)
      END [PARAMETERS FOR PRESTO REGION
        <string>region_name
    END [TIME STEPPING BLOCK <string>time_block_name]

    TERMINATION TIME = <real>termination_time

  END TIME CONTROL

# Begin Region scope

  BEGIN PRESTO REGION <string>region_specification

    USE FINITE ELEMENT MODEL <string>model_name

    # Boundary conditions

    BEGIN FIXED DISPLACEMENT
      SURFACE = <string list>surface_names |
        NODE SET = <string list>nodelist_names
      COMPONENTS = <string>X/Y/Z
    END [FIXED DISPLACEMENT]

    BEGIN PRESCRIBED DISPLACEMENT
      SURFACE = <string list>surface_names |
        NODE SET = <string list>nodelist_names
      DIRECTION = <string>defined_direction |
        COMPONENT = <string>X|Y|Z
      FUNCTION = <string>function_name
      SCALE FACTOR = <real>scale_factor(1.0)
    END [PRESCRIBED DISPLACEMENT]

    BEGIN PRESCRIBED VELOCITY
      SURFACE = <string list>surface_names |
        NODE SET = <string list>nodelist_names
      DIRECTION = <string>defined_direction |
        COMPONENT = <string>X|Y|Z |
        CYLINDRICAL AXIS = <string>defined_axis |
        RADIAL AXIS = <string>defined_axis
      FUNCTION = <string>function_name
```

```
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED VELOCITY]

BEGIN PRESCRIBED ACCELERATION
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED ACCELERATION]

BEGIN FIXED ROTATION
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  COMPONENTS = <string>X/Y/Z
END [FIXED ROTATION]

BEGIN PRESCRIBED ROTATION
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED ROTATION]

BEGIN INITIAL VELOCITY
  BLOCK = <string list>block_names
  DIRECTION = <string>defined_direction
  MAGNITUDE = <real>magnitude_of_velocity
END [INITIAL VELOCITY]

BEGIN INITIAL VELOCITY
  BLOCK = <string list>block_names
  CYLINDRICAL AXIS = <string>defined_cylindrical_axis
  ANGULAR VELOCITY = <real>angular_velocity
END [INITIAL VELOCITY]

BEGIN PRESSURE
  SURFACE = <string list>surface_names
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESSURE]

BEGIN PRESCRIBED FORCE
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED FORCE]

BEGIN PRESCRIBED MOMENT
  SURFACE = <string list>surface_names |
    NODE SET = <string list>nodelist_names
  DIRECTION = <string>defined_direction |
```

```
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
END [PRESCRIBED MOMENT]

# Specialized boundary conditions

BEGIN GRAVITY
  DIRECTION = <string>defined_direction
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
  GRAVITATIONAL CONSTANT = <real>g_constant
END [GRAVITY]

BEGIN CAVITY EXPANSION
  SURFACE = <string>surface_name
  EXPANSION RADIUS = <string>spherical|
    cylindrical(spherical)
  TARGET NORMAL = <string>X|Y|Z
  FREE SURFACE = <real>upper_surface
    <real>lower_surface
  LAYER SURFACE = <real>upper_layer <real>lower_layer
  PRESSURE COEFFICIENTS =
    <real>coeff1 <real>coeff2 <real>coeff3
  NODE SETS TO DEFINE BODY AXIS =
    <string>nodelist_id1 <string>nodelist_id2
  TIP RADIUS = <real>tip_radius
END [CAVITY EXPANSION]

BEGIN PERIODIC
  NODE SETS = <string>nodelist_id1
    <string>nodelist_id2 |
    SURFACES = <string>surface_id1
      <string>surface_id2
  SEARCH TOLERANCE = <real>search_tolerance(1.0e-4)
  PRESCRIBED QUANTITY =
    <string>DISPLACEMENT|VELOCITY|FORCE
  COMPONENT  = <string>X|Y|Z  |
    RADIAL AXIS = <string>defined_radial_axis
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor(1.0)
  THETA = <real>theta_value>
END [PERIODIC]

BEGIN SILENT BOUNDARY
  SURFACE = <string>surface_name
END [SILENT BOUNDARY]

BEGIN SPOT WELD
  NODE SET = <string>nodelist_id
  SURFACE = <surface>surface_id
  NORMAL DISPLACEMENT FUNCTION =
    <string>function_nor_disp
  NORMAL DISPLACEMENT SCALE FACTOR =
    <real>scale_nor_disp
  TANGENTIAL DISPLACEMENT FUNCTION =
    <string>function_tang_disp
  TANGENTIAL DISPLACEMENT SCALE FACTOR =
```

```
    <real>scale_tang_disp
  FAILURE ENVELOPE EXPONENT = <real>exponent
  FAILURE DECAY CYCLES = <integer>number_decay_cycles
END [SPOT WELD]

# Constraints

BEGIN CONSTRAINT CONSTANT DISTANCE
  NODE SETS = <string list>nodelist_id1 nodelist_id2
  SEARCH TOLERANCE = <real>search_tol(0.0001)
END [CONSTRAINT CONSTANT DISTANCE]

BEGIN CONSTRAINT HEX SHELL <string>constraint_name
  HEX SHELL BLOCK = <string>hexshell_block_id
  EXCLUDE SURFACE = <string>surface_id
END [CONSTRAINT HEX SHELL <string>contraint_name]

# Mass property calculations

BEGIN MASS PROPERTIES
  BLOCK = <string>block_id1 <string>block_id2 ...
  STRUCTURE NAME = <string>structure_name
END [MASS PROPERTIES]

# Contact

BEGIN CONTACT DEFINITION <string>block_name
  CONTACT SURFACE <string>name
    CONTAINS <string list>surfaces
  CONTACT ALL BLOCKS
  REMOVE INITIAL OVERLAP
  MULTIPLE INTERACTIONS WITH ANGLE
    <real>angle(60.0)
  NUMBER OF ITERATIONS
    <integer>number_enforce_iter(5)
  BEGIN FRICTIONLESS MODEL <string>name
  END [FRICTIONLESS MODEL <string>name]
  BEGIN CONSTANT FRICTION MODEL <string>name
    FRICTION COEFFICIENT = <real>coeff
  END [CONTACT FRICTION MODEL <string>name]
  BEGIN TIED MODEL <string>name
  END [TIED MODEL <string>name]
  BEGIN ANALYTIC PLANE [<string>name]
    NORMAL = <string>defined_direction
    POINT = <string>defined_point
  END [ANALYTIC PLANE <string>name]
  BEGIN ANALYTIC CYLINDER [<string>name]
    CENTER = <string>defined_point
    AXIAL DIRECTION = <string>defined_axis
    RADIUS = <real>cylinder_radius
    LENGTH = <real>cylinder_length
    CONTACT NORMAL = <string>OUTSIDE|INSIDE
  END [ANALYTIC CYLINDER <string>name]
  BEGIN ANALYTIC SPHERE [<string>name]
    CENTER = <string>defined_point
    RADIUS = <real>sphere_radius
  END [ANALYTIC SPHERE <string>name]
  BEGIN DEFAULTS [<string>name]
```

```
    NORMAL TOLERANCE = <real>norm_tol
    TANGENTIAL TOLERANCE = <real>tang_tol
    OVERLAP NORMAL TOLERANCE = <real>norm_tol
    OVERLAP TANGENTIAL TOLERANCE = <real>tang_tol
    FRICTION MODEL = <string>name
    AUTOMATIC KINEMATIC PARTITION
  END [DEFAULTS <string>name]
  BEGIN INTERACTION [<string>name]
    MASTER = <string>surface
    SLAVE = <string>surface
    SURFACES = <string>surface1 <string>surface2
    KINEMATIC PARTITION = <real>kin_part
    NORMAL TOLERANCE = <real>norm_tol
    TANGENTIAL TOLERANCE = <real>tang_tol
    OVERLAP NORMAL TOLERANCE = <real>norm_tol
    OVERLAP TANGENTIAL TOLERANCE = <real>tang_tol
    FRICTION MODEL = <string>name
    AUTOMATIC KINEMATIC PARTITION
  END [INTERACTION <string>name]
END [CONTACT DEFINITION <string>block_name]

# Results specification

BEGIN RESULTS OUTPUT <string>results_name
  DATABASE NAME = <string>results_file_name
  DATABASE TYPE =
    <string>database_type(exodusII)
  NODE VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  NODAL VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  ELEMENT VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  GLOBAL VARIABLES  = <string>variable_name
    [AS <string>dbase_variable_name
    <string>variable_name AS
    <string>dbase_variable_name ...]
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
END [RESULTS OUTPUT <string>results_name]

# History specification
```

```
      BEGIN HISTORY OUTPUT <string>history_name
        DATABASE NAME = <string>history_file_name
        DATABASE TYPE =
          <string>database_type(exodusII)
        VARIABLE = <string>entity_type
          <string>internal_name
          AT <string>entity_type <integer>entity_id
          [AS <string>history_variable_name]
        VARIABLE = <string>entity_type_global
          <string>internal_name
          [AS <string>history_variable_name]
        START TIME = <real>output_start_time
        TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
        AT TIME <real>time_begin INCREMENT =
          <real>time_increment_dt
        ADDITIONAL TIMES = <real>output_time1
          <real>output_time2 ...
        AT STEP <integer>step_begin INCREMENT =
          <integer>step_increment
        ADDITIONAL STEPS = <integer>output_step1
          <integer>output_step2 ...
        TERMINATION TIME = <real>termination_time_value
      END [HISTORY OUTPUT <string>history_name]

      # Restart specification

      BEGIN RESTART DATA <string>restart_name
        DATABASE NAME = <string>restart_file
        INPUT DATABASE NAME = <string>restart_input_file
        OUTPUT DATABASE NAME =
          <string>restart_output_file
        DATABASE TYPE =
          <string>database_type(exodusII)
        START TIME = <real>restart_start_time
        TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
        AT TIME <real>time_begin INCREMENT =
          <real>time_increment_dt
        ADDITIONAL TIMES = <real>output_time1
          <real>output_time2 ...
        AT STEP <integer>step_begin INCREMENT =
          <integer>step_increment
        ADDITIONAL STEPS = <integer>output_step1
          <integer>output_step2 ...
        TERMINATION TIME = <real>termination_time_value
      END [RESTART DATA <string>restart_name]

    END [PRESTO REGION <string>region_name]

  END [PRESTO PROCEDURE <string>procedure_name]

END [SIERRA <string>name]
```

# Appendix B: Registered Variables

This appendix contains the registered variables that the user can select as output to the results file. The registered variables are presented in tables based on use, as follows:

- Table B.1 Variables Registered on Nodes (Variable and Type)
- Table B.2 Element Variables Registered for Energy-Dependent ("Equation-of-State") Elements
- Table B.3 Element Variables Registered for Solid Elements
- Table B.4 Element Variables Registered for Membranes
- Table B.5 Nodal Variables Registered for Shells
- Table B.6 Element Variables Registered for Shells
- Table B.7 Element Variables Registered for Truss
- Table B.8 Global Registered Variables

The tables provide the following information about each registered variable:

**Variable Name.** This is the string that will appear on the `NODE VARIABLES` or `ELEMENT VARIABLES` command line.

**Type.** This is the variable's type. The types are `tReal`, `tVec3D`, `tSym33`, and `tFull36`. The type `tReal` indicates the registered variable is a real. The type `tVec3D` indicates the registered variable is a three-dimensional vector. For a three-dimensional vector, the variable quantities will be output with suffixes of `_X`, `_Y`, and `_Z`. For example, if the registered variable `displacement` is requested to be output as `displ`, the components of the displacement vector on the results file will be `displ_x`, `displ_y`, and `displ_z`. The type `tSym33` (or tSymTen33) indicates the registered variable is a symmetric $3 \times 3$ tensor. For a $3 \times 3$ symmetric tensor, the variable quantities will be output with suffixes of `_xx`, `_yy`, `_zz`, `_xy`, `_yz`, and `_zx`. For example, if the registered variable `rotated_stress` is requested for output as `stress`, the components of the stress tensor on the results file will be `stress_xx`, `stress_yy`, `stress_zz`, `stress_xy`, `stress_yz`, and `stress_zx`. The type `tFull36` is a full $3 \times 3$ tensor with three diagonal terms and six off-diagonal terms.

**State Specification.** It is possible to output the state variables from the material models, but the implementation of this feature is not complete at this time. Work is under way to improve the method for output of the state variables. For the elastic material model, there are no state variables. The state variables for the orthotropic crush model and the energy-dependent models (Mie-Gruneisen, Mie-Gruneisen Power-Series, JWL, and ideal gas) are accessed as any other registered variables. To get the state variables for the elastic-plastic (`elastic_plastic`), elastic-plastic power-law hardening (`ep_power_hard`), foam plasticity (`foam_plasticity`), and orthotropic rate (`orthotropic_rate`) material models, you should use the

`ELEMENT VARIABLES` command line in the `RESULTS OUTPUT` command block in the form

```
ELEMENT VARIABLES = state_variables_"material_id" .
```

For example, if you wanted to get the state variables for the elastic-plastic power-law hardening material model, you would use

```
ELEMENT VARIABLES = state_variables_ep_power_hard .
```

The above command will output all of the state variables for the elastic-plastic power-law hardening material model. You will have to consult with William Scherzinger (wmscher@sandia.gov) to get the current ordering for the state variables.

For more information about the output of state variables, contact Richard Koteras (jrkoter@sandia.gov), Arne Gullerud (asgulle@sandia.gov), or William Scherzinger (wmscher@sandia.gov).

**Comments.** Additional information may be provided to help the user select the registered variable.

The tables of registered variables follow.

### Table B.1. Variables Registered on Nodes (Variable and Type)

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| reactions | tVec3D | temporary | |
| moment_reactions | tVec3D | temporary | |
| model_coordinates | tVec3D | model | |
| coordinates | tVec3D | temporary | |
| displacement | tVec3D | state | |
| displacement_increment | tVec3D | temporary | |
| velocity | tVec3D | state | |
| acceleration | tVec3D | state | |
| force_internal | tVec3D | state | |
| force_external | tVec3D | state | |
| force_hourglass | tVec3D | state | |
| mass | tReal | model | |
| force_contact | tVec3D | state | |

**Table B.1. Variables Registered on Nodes (Variable and Type) (Continued)**

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| moment_reactions | tVec3D | temporary | |

**Table B.2. Element Variables Registered for Energy-Dependent ("Equation-of-State") Elements**

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| stress | tSym33 | state | |
| stretch | tSym33 | persistent | |
| rotation | tFull36 | state | |
| element_density | Real | state | |
| sound_speed | tReal | state | |
| specific_internal_energy | tReal | state | |
| artificial_viscosity | tReal | state | |
| element_mass | tReal | model | |
| volume | tReal | temporary | |
| shrmod | tReal | temporary | |
| dilmod | tReal | temporary | |
| rotated_stress | tSym33 | temporary | |
| stress | tSym33 | state | |

**Table B.3. Element Variables Registered for Solid Elements**

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| stress | tSym33 | state | |
| stretch | tSym33 | persistent | |
| rotation | tFull36 | state | |

### Table B.3. Element Variables Registered for Solid Elements (Continued)

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| element_mass | tReal | model | |
| volume | tReal | temporary | |
| shrmod | tReal | temporary | |
| dilmod | tReal | temporary | |
| rotated_stress | tSym33 | temporary | |
| stress | tSym33 | state | |

### Table B.4. Element Variables Registered for Membranes

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| memb_stress | tSym33 | temporary | |
| element_area | tReal | state | |
| element_thickness | tReal | state | |
| element_mass | tReal | model | |

### Table B.5. Nodal Variables Registered for Shells

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| rotational_displacement | tVec3D | state | |
| rotational_velocity | tVec3D | state | |
| rotational_acceleration | tVec3D | state | |
| moment_internal | tVec3D | state | |
| moment_external | tVec3D | state | |
| rotational_mass | tReal | temporary | |

### Table B.6. Element Variables Registered for Shells

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| memb_stress | tSymTen33 | temporary | |
| bottom_stress | tSymTen33 | temporary | |
| top_stress | tSymTen33 | temporary | |
| element_area | tReal | state | |
| element_thickness | tReal | state | |
| element_mass | tReal | model | |

### Table B.7. Element Variables Registered for Truss

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| truss_init_length | tReal | model | |
| truss_stretch | tReal | persistent | |
| truss_stress | tReal | state | |
| truss_strain_incr | tReal | state | |
| element_mass | tReal | model | |

### Table B.8. Global Registered Variables

| Variable Name | Type | State Specification | Comments |
|---|---|---|---|
| timestep | tReal | temporary | |
| KineticEnergy | tReal | temporary | |

# Index

This index lists the command blocks and command lines in Presto. In general, single-level entries identify the page where the command syntax appears, with discussion following soon thereafter—on the same page or on a subsequent page. Page ranges are not provided in this index. Some entries consist of two or more levels. Such entries are typically based on context, including such information as the command blocks in which a command line appears, the location of the discussion related to a particular command line, and tips on usage. The PDF version of this document contains hyperlinked entries from the page numbers listed in this index to the text in the body of the document. Information in appendices is not included in this index.


**A**

A
  in JWL material model 46
A0
  in Soil and Crushable Foam material model 36
A1
  in Soil and Crushable Foam material model 36
A2
  in Soil and Crushable Foam material model 36
ABSCISSA 29
ADDITIONAL STEPS
  in History Output 100
    description of 103
  in Restart Data 104
    description of 106
  in Results Output 95
    description of 99
ADDITIONAL TIMES
  in History Output 100
    description of 102
  in Restart Data 104
    description of 106
  in Results Output 95
    description of 98
ALIAS 49
ANALYTIC CYLINDER
  in Contact Definition 82
    about 86
    description of 87
ANALYTIC PLANE
  in Contact Definition 82
    about 86
    description of 86
ANALYTIC SPHERE
  in Contact Definition 82
    about 86
    description of 88
ANGULAR VELOCITY
  in Initial Velocity
    for angular velocity 67
AT STEP
  in History Output 100
    description of 103
  in Restart Data 104
    description of 106
  in Results Output 95
    description of 98
AT TIME
  in History Output 100
    description of 102
  in Restart Data 104
    description of 106
  in Results Output 95
    description of 98
AUTOMATIC KINEMATIC PARTITION
  in Contact Definition
    in Defaults 82, 88
      description of 91
    in Interaction 83, 91
AXIAL DIRECTION
  in Contact Definition
    in Analytic Cylinder 82
      description of 87

**B**

B
  in JWL material model 46
B5
  in JWL material model 46
BETA
  in Elastic-Plastic material model 34
  in Foam Plasticity material model 38
BLOCK 80
  in Initial Velocity 67
    for angular velocity 67
    for direction 67
BULK MODULUS
  in Elastic material model 33
  in Elastic-Plastic material model 34
  in Elastic-Plastic Power-Law Hardening material model 35
  in Foam Plasticity material model 38
  in Orthotropic Crush material model 40
  in Orthotropic Rate material model 42
  in Soil and Crushable Foam material model 36

**C**

C_0
in Ideal Gas material model 47
in Mie-Gruneisen material model 44
in Mie-Gruneisen Power-Series material model 45
CAVITY EXPANSION 71
CENTER
in Contact Definition
in Analytic Cylinder 82
description of 87
in Analytic Sphere 82
description of 88
COMPONENT
for Periodic boundary condition 73
for Prescribed Acceleration boundary condition 65
for Prescribed Displacement boundary condition 63
for Prescribed Force boundary condition 69
for Prescribed Moment boundary condition 70
for Prescribed Rotation boundary condition 66
for Prescribed Velocity boundary condition 64
COMPONENTS
for Fixed Displacement boundary condition 63
for Fixed Rotation boundary condition 65
CONSTANT FRICTION MODEL
in Contact Definition 82
about 85
description of 86
CONSTRAINT CONSTANT DISTANCE 78
CONSTRAINT HEX SHELL 78
CONTACT ALL BLOCKS
in Contact Definition 82
description of 84
use of 83
CONTACT DEFINITION 82
example of 93
use of 83
CONTACT NORMAL
in Contact Definition
in Analytic Cylinder 82
description of 87
CONTACT SURFACE
in Contact Definition 82
description of 83
use of 83, 84
CRUSH XX
in Orthotropic Crush material model 40
CRUSH XY
in Orthotropic Crush material model 40
CRUSH YY
in Orthotropic Crush material model 40
CRUSH YZ
in Orthotropic Crush material model 40
CRUSH ZX
in Orthotropic Crush material model 40
CRUSH ZZ
in Orthotropic Crush material model 40
CYLINDRICAL AXIS
for Prescribed Velocity boundary condition 64
in Initial Velocity
for angular velocity 67

**D**

D
in JWL material model 46
DAMPER AREA 54
DATABASE NAME 49
in History Output 100
in Restart Data 104
in Results Output 95
DATABASE TYPE 49
in History Output 100
in Restart Data 104
in Results Output 95
DEACTIVATE 56
DEFAULTS
in Contact Definition 82
about 81
description of 88
DEFINE AXIS WITH POINT DIRECTION 30
DEFINE AXIS WITH POINT POINT 30
DEFINE DIRECTION 30
DEFINE POINT 15, 30
DEFINITION FOR FUNCTION 29
DENSITY
in Elastic material model 33
in Elastic-Plastic material model 34
in Elastic-Plastic Power-Law Hardening material model 35
in Foam Plasticity material model 38
in Orthotropic Crush material model 40
in Orthotropic Rate material model 42
in Soil and Crushable Foam material model 36
DEPOSIT SPECIFIC INTERNAL ENERGY 54
DIRECTION
for Gravity boundary condition 71
for Prescribed Acceleration boundary condition 65
for Prescribed Displacement boundary condition 63
for Prescribed Force boundary condition 69
for Prescribed Moment boundary condition 70
for Prescribed Rotation boundary condition 66
for Prescribed Velocity boundary condition 64
in Initial Velocity
for direction 67

**E**

E_0
in JWL material model 46
ELEMENT NUMERICAL FORMULATION 55
ELEMENT STRAIN FORMULATION 52
ELEMENT VARIABLES
in Results Output 95
description of 97
EX
in Orthotropic Crush material model 40
EXCLUDE SURFACE 78
EXPANSION RADIUS
for Cavity Expansion boundary condition 71
EY
in Orthotropic Crush material model 40
EZ
in Orthotropic Crush material model 40

**F**
FAILURE DECAY CYCLES
for Spot-Weld boundary condition 75
FAILURE ENVELOPE EXPONENT
for Spot-Weld boundary condition 75
FINITE ELEMENT MODEL 49
FIXED DISPLACEMENT 63
FIXED ROTATION 65
FREE SURFACE
for Cavity Expansion boundary condition 71
FREE SURFACE EFFECT COEFFICIENTS
for Cavity Expansion boundary condition 71
FRICTION COEFFICIENT
in Contact Definition
in Constant Friction Model 82
description of 86
FRICTION MODEL
in Contact Definition
in Defaults 82, 88
description of 90
in Interaction 82, 91
FRICTIONLESS MODEL
in Contact Definition 82
about 85
description of 86
FUNCTION
for Gravity boundary condition 71
for Periodic boundary condition 73
for Prescribed Acceleration boundary condition 65
for Prescribed Displacement boundary condition 63
for Prescribed Force boundary condition 69
for Prescribed Moment boundary condition 70
for Prescribed Rotation boundary condition 66
for Prescribed Velocity boundary condition 64
for Pressure boundary condition 69

**G**
GAMMA
in Ideal Gas material model 47
GAMMA_0
in Mie-Gruneisen material model 44
in Mie-Gruneisen Power-Series material model 45
GLOBAL VARIABLES
in Results Output 95
description of 97
GRAVITATIONAL CONSTANT
for Gravity boundary condition 71
GRAVITY 71
GXY
in Orthotropic Crush material model 40
GYZ
in Orthotropic Crush material model 40
GZX
in Orthotropic Crush material model 40

**H**
HARDENING CONSTANT
in Elastic-Plastic Power-Law Hardening material model 35
HARDENING EXPONENT
in Elastic-Plastic Power-Law Hardening material model 35
HARDENING MODULUS
in Elastic-Plastic material model 34
HEX HOURGLASS STIFFNESS 52
HEX HOURGLASS VISCOSITY 52
HEX SHELL BLOCK 78
HISTORY OUTPUT 100
HYDRO EXPONENT
in Foam Plasticity material model 38
HYDRO HARDENING
in Foam Plasticity material model 38
HYDRO STRENGTH
in Foam Plasticity material model 38

**I**
INITIAL TIME STEP 59
in Parameters for Presto Region 59
INITIAL VELOCITY
about 67
for angular velocity 67
for direction 67
INPUT DATABASE NAME
in Restart Data 104
INTERACTION
in Contact Definition 82
about 81
description of 91

**K**
K1
in Mie-Gruneisen Power-Series material model 45
K2
in Mie-Gruneisen Power-Series material model 45
K3
in Mie-Gruneisen Power-Series material model 45
K4
in Mie-Gruneisen Power-Series material model 45
K5
in Mie-Gruneisen Power-Series material model 45
KINEMATIC PARTITION
in Contact Definition
in Interaction 82, 91
description of 93

**L**
L FUNCTION
in Orthotropic Rate material model 42
LAMBDA
in Elastic material model 33
in Elastic-Plastic material model 34
in Elastic-Plastic Power-Law Hardening material model 35
in Foam Plasticity material model 38
in Orthotropic Crush material model 40
in Orthotropic Rate material model 42
in Soil and Crushable Foam material model 36
LAYER SURFACE
for Cavity Expansion boundary condition 71
LENGTH

in Contact Definition
in Analytic Cylinder 82
description of 87
LINEAR BULK VISCOSITY 52
LUDERS STRAIN
in Elastic-Plastic Power-Law Hardening material model 35
LW FUNCTION
in Orthotropic Rate material model 42
LX
in Orthotropic Rate material model 42
LY
in Orthotropic Rate material model 42
LZ
in Orthotropic Rate material model 42

**M**

MAGNITUDE
in Initial Velocity
for direction 67
MASS PROPERTIES 80
MASTER
in Contact Definition
in Interaction 82, 91
description of 92
MATERIAL 51
MEMBRANE SCALE THICKNESS 52
MODULUS FUNCTION
in Orthotropic Rate material model 42
MODULUS LLLL
in Orthotropic Rate material model 42
MODULUS LLWW
in Orthotropic Rate material model 42
MODULUS LWLW
in Orthotropic Rate material model 42
MODULUS TLTL
in Orthotropic Rate material model 42
MODULUS TTLL
in Orthotropic Rate material model 42
MODULUS TTTT
in Orthotropic Rate material model 42
MODULUS TTWW
in Orthotropic Rate material model 42
MODULUS WTWT
in Orthotropic Rate material model 42
MODULUS WWWW
in Orthotropic Rate material model 42
MULTIPLE INTERACTIONS WITH ANGLE
in Contact Definition 82
description of 85

**N**

NODAL VARIABLES
in Results Output 95
description of 96
NODE SET
for Fixed Displacement boundary condition 63
for Fixed Rotation boundary condition 65
for Prescribed Acceleration boundary condition 65
for Prescribed Displacement boundary condition 63
for Prescribed Force boundary condition 69
for Prescribed Moment boundary condition 70
for Prescribed Rotation boundary condition 66
for Prescribed Velocity boundary condition 64
for Spot-Weld boundary condition 75
NODE SETS 78
for Periodic boundary condition 73
NODE SETS TO DEFINE BODY AXIS
for Cavity Expansion boundary condition 71
NODE VARIABLES
in Results Output 95
description of 96
NORMAL
in Contact Definition
in Analytic Plane 82
description of 86
NORMAL DISPLACEMENT FUNCTION
for Spot-Weld boundary condition 75
NORMAL DISPLACEMENT SCALE FACTOR
for Spot-Weld boundary condition 75
NORMAL TOLERANCE
in Contact Definition
in Defaults 82, 88
description of 89
in Interaction 82, 91
NUMBER OF ITERATIONS
in Contact Definition 82
description of 85

**O**

OMEGA
in JWL material model 46
ORDINATE 29
OUTPUT DATABASE NAME
in Restart Data 104
OVERLAP NORMAL TOLERANCE
in Contact Definition
in Defaults 82, 88
description of 90
in Interaction 82, 91
OVERLAP TANGENTIAL TOLERANCE
in Contact Definition
in Defaults 82, 88
description of 90
in Interaction 82, 91

**P**

PARAMETERS FOR BLOCK 49
about 50
PARAMETERS FOR MODEL ELASTIC
in Elastic material model 33
PARAMETERS FOR MODEL ELASTIC PLASTIC
in Elastic-Plastic material model 34
PARAMETERS FOR MODEL EP_POWER_HARD
in Elastic-Plastic Power-Law Hardening material model 35
PARAMETERS FOR MODEL FOAM_PLASTICITY
in Foam Plasticity material model 38
PARAMETERS FOR MODEL IDEAL_GAS
in Ideal Gas material model 47

PARAMETERS FOR MODEL JWL
in JWL material model 46
PARAMETERS FOR MODEL MIE_GRUNEISEN 44
PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES
in Mie-Gruneisen Power-Series material model 45
PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
in Orthotropic Crush material model 40
PARAMETERS FOR MODEL ORTHOTROPIC_RATE
in Orthotropic Rate material model 42
PARAMETERS FOR MODEL SOIL_FOAM
in Soil and Crushable Foam material model 36
PARAMETERS FOR PRESTO REGION
in Time Stepping Block 57
contents of 59
PERIODIC 73
PHI
in Foam Plasticity material model 38
PMIN
in Mie-Gruneisen material model 44
in Mie-Gruneisen Power-Series material model 45
POINT
in Contact Definition
in Analytic Plane 82
description of 86
POISSONS RATIO
in Elastic material model 33
in Elastic-Plastic material model 34
in Elastic-Plastic Power-Law Hardening material model 35
in Foam Plasticity material model 38
in Orthotropic Crush material model 40
in Orthotropic Rate material model 42
in Soil and Crushable Foam material model 36
POISSR
in Mie-Gruneisen material model 44
in Mie-Gruneisen Power-Series material model 45
PRESCRIBED ACCELERATION 65
PRESCRIBED DISPLACEMENT 63
PRESCRIBED FORCE 69
PRESCRIBED MOMENT 70
PRESCRIBED QUANTITY
for Periodic boundary condition 73
PRESCRIBED ROTATION 66
PRESCRIBED VELOCITY 64
PRESSURE 69
PRESSURE COEFFICIENTS
for Cavity Expansion boundary condition 71
PRESSURE CUTOFF
in Soil and Crushable Foam material model 36
PRESSURE FUNCTION
in Soil and Crushable Foam material model 36
PRESTO PROCEDURE 57
about 57
PRESTO REGION 61
about 57
PROPERTY SPECIFICATION FOR MATERIAL 32
about 51

**Q**

QUADRATIC BULK VISCOSITY 52

**R**

R1
in JWL material model 46
R2
in JWL material model 46
RADIAL AXIS
for Periodic boundary condition 73
for Prescribed Velocity boundary condition 64
RADIUS
in Contact Definition
in Analytic Cylinder 82
description of 87
in Analytic Sphere 82
description of 88
RATE FUNCTION
in Orthotropic Rate material model 42
REMOVE INITIAL OVERLAP
in Contact Definition 82
description of 84
RESTART 28
about 26
RESTART DATA 104
about 26
RESTART TIME 27
about 26
with Restart Data 104
RESULTS OUTPUT 95
RHO_0
in Ideal Gas material model 47
in JWL material model 46
in Mie-Gruneisen material model 44
in Mie-Gruneisen Power-Series material model 45

**S**

SCALE FACTOR
for Gravity boundary condition 71
for Periodic boundary condition 73
for Prescribed Acceleration boundary condition 65
for Prescribed Displacement boundary condition 63
for Prescribed Force boundary condition 69
for Prescribed Moment boundary condition 70
for Prescribed Rotation boundary condition 66
for Prescribed Velocity boundary condition 64
for Pressure boundary condition 69
SEARCH TOLERANCE 78
for Periodic boundary condition 73
SHEAR EXPONENT
in Foam Plasticity material model 38
SHEAR HARDENING
in Foam Plasticity material model 38
SHEAR MODULUS
in Elastic material model 33
in Elastic-Plastic material model 34
in Elastic-Plastic Power-Law Hardening material model 35
in Foam Plasticity material model 38
in Orthotropic Crush material model 40
in Orthotropic Rate material model 42

in Soil and Crushable Foam material model 36
SHEAR STRENGTH
in Foam Plasticity material model 38
SHELL INTEGRATION POINTS 53
SHELL INTEGRATION SCHEME 53
SHELL SCALE THICKNESS 53
SHUG
in Mie-Gruneisen material model 44
SIERRA 26
SILENT BOUNDARY 75
SLAVE
in Contact Definition
in Interaction 82, 91
description of 92
SOLID MECHANICS USE MODEL 51
SPOT WELD 75
START TIME 57
in History Output 100
description of 102
in Restart Data 104
description of 105
in Results Output 95
description of 98
STEP INTERVAL 60
in Parameters for Presto Region 59
STRUCTURE NAME 80
SURFACE
for Cavity Expansion boundary condition 71
for Fixed Displacement boundary condition 63
for Fixed Rotation boundary condition 65
for Prescribed Acceleration boundary condition 65
for Prescribed Displacement boundary condition 63
for Prescribed Force boundary condition 69
for Prescribed Moment boundary condition 70
for Prescribed Rotation boundary condition 66
for Prescribed Velocity boundary condition 64
for Pressure boundary condition 69
for Spot-Weld boundary condition 75
in Silent boundary condition 75
SURFACES
for Periodic boundary condition 73
in Contact Definition
in Interaction 82, 91
description of 92

**T**

T FUNCTION
in Orthotropic Rate material model 42
TANGENTIAL DISPLACEMENT FUNCTION
for Spot-Weld boundary condition 75
TANGENTIAL DISPLACEMENT SCALE FACTOR
for Spot-Weld boundary condition 75
TANGENTIAL TOLERANCE
in Contact Definition
in Defaults 82, 88
description of 89
in Interaction 82, 91
TARGET NORMAL
for Cavity Expansion boundary condition 71
TDET
in JWL material model 46
TERMINATION TIME 58
in History Output 100
description of 103
in Restart Data 104
description of 106
in Results Output 95
description of 99
THETA
for Periodic boundary condition 73
TIED MODEL
in Contact Definition 82
about 85
description of 86
TIME CONTROL 57
about 57
example of 61
TIME STEP INCREASE FACTOR 59
in Parameters for Presto Region 59
TIME STEP SCALE FACTOR 59
in Parameters for Presto Region 59
with Element Numerical Formulation 55
TIME STEPPING BLOCK 57
TIMESTEP ADJUSTMENT INTERVAL
in History Output 100
description of 102
in Restart Data 104
description of 105
in Results Output 95
description of 98
TIP RADIUS
for Cavity Expansion boundary condition 71
TITLE 26
TL FUNCTION
in Orthotropic Rate material model 42
TRUSS AREA 54
TX
in Orthotropic Rate material model 42
TY
in Orthotropic Rate material model 42
TYPE 29
TZ
in Orthotropic Rate material model 42

**U**

USE FINITE ELEMENT MODEL 62

**V**

VALUES 29
VARIABLE
in History Output
about 101
for global variables 100
description of 101
for nodal and element variables 100
description of 101
VMIN
in Orthotropic Crush material model 40

**W**

W FUNCTION
in Orthotropic Rate material model 42
WT FUNCTION
in Orthotropic Rate material model 42

**X**

XDET
in JWL material model 46

**Y**

Y_0
in Mie-Gruneisen material model 44
in Mie-Gruneisen Power-Series material model 45
YDET
in JWL material model 46
YIELD STRESS
in Elastic-Plastic material model 34
in Elastic-Plastic Power-Law Hardening material model 35
in Orthotropic Crush material model 40
in Orthotropic Rate material model 42
YOUNGS MODULUS
in Elastic material model 33
in Elastic-Plastic material model 34
in Elastic-Plastic Power-Law Hardening material model 35
in Foam Plasticity material model 38
in Orthotropic Crush material model 40
in Orthotropic Rate material model 42
in Soil and Crushable Foam material model 36

**Z**

ZDET
in JWL material model 46

# Distribution

Scott W. Doebling
Los Alamos National Laboratory
PO Box 1663 MS P946
Los Alamos, NM 87545

### Internal

| | | |
|---|---|---|
| 1 | MS-0318 | P. Yarrington, 9230 |
| 1 | MS-0321 | W. J. Camp, 9200 |
| 1 | MS-0427 | J. R. Weatherby, 2134 |
| 1 | MS-0521 | S. T. Montgomery, 2561 |
| 1 | MS-0807 | B. Cole, 9338 |
| 1 | MS-0819 | K. H. Brown, 9231 |
| 1 | MS-0819 | D. E. Carroll, 9231 |
| 1 | MS-0819 | S. Carroll, 9231 |
| 1 | MS-0819 | R. R. Drake, 9231 |
| 1 | MS-0819 | R. M. Summers, 9231 |
| 1 | MS-0826 | J. S. Rath, 9143 |
| 1 | MS-0826 | J. R. Stewart, 9143 |
| 1 | MS-0826 | J. D. Zepper, 9143 |
| 1 | MS-0827 | H. C. Edwards, 9143 |
| 1 | MS-0827 | M. E. Hamilton, 9143 |
| 1 | MS-0827 | T. J. Otahal, 9143 |
| 1 | MS-0834 | A. C. Ratzel, 9110 |
| 1 | MS-0834 | P. R. Schunk, 9114 |
| 1 | MS-0835 | K. F. Alvin, 9142 |
| 1 | MS-0835 | E. A. Boucheron, 9141 |
| 1 | MS-0835 | S. W. Bova, 9141 |
| 1 | MS-0835 | N. K. Crane, 9142 |
| 1 | MS-0835 | R. R. Lober, 9141 |
| 1 | MS-0835 | J. M. McGlaun, 9140 |
| 1 | MS-0835 | K. H. Pierson, 9142 |
| 1 | MS-0835 | T. F. Walsh, 9142 |
| 1 | MS-0841 | T. C. Bickel, 9100 |
| 1 | MS-0847 | C. R. Adams, 9125 |
| 1 | MS-0847 | S. W. Attaway, 9134 |
| 1 | MS-0847 | M. K. Bhardwaj, 9142 |
| 1 | MS-0847 | F. Bitsie, 9124 |
| 1 | MS-0847 | M. L. Blanford, 9142 |
| 1 | MS-0847 | S. N. Burchett, 9126 |
| 1 | MS-0847 | H. Duong, 9126 |

| | | |
|---|---|---|
| 1 | MS-0847 | C. W. Fulcher, 9125 |
| 1 | MS-0847 | K. W. Gwinn, 9126 |
| 10 | MS-0847 | A. S. Gullerud, 9142 |
| 1 | MS-0847 | T. D. Hinnerichs, 9126 |
| 1 | MS-0847 | J. Jung, 9127 |
| 1 | MS-0847 | S. W. Key, 9142 |
| 10 | MS-0847 | J. R. Koteras, 9142 |
| 1 | MS-0847 | J. S. Lash, 9126 |
| 1 | MS-0847 | R. A. May, 9126 |
| 1 | MS-0847 | K. E. Metzinger, 9126 |
| 1 | MS-0847 | J. A. Mitchell, 9142 |
| 1 | MS-0847 | H. S. Morgan, 9120 |
| 1 | MS-0847 | V. L. Porter, 9142 |
| 1 | MS-0847 | G. M. Reese, 9142 |
| 1 | MS-0847 | G. D. Sjaardema, 9143 |
| 1 | MS-0847 | C. M. Stone, 9142 |
| 1 | MS-0847 | J. W. Swegle, 9142 |
| 1 | MS-0893 | M. K. Neilsen, 9123 |
| 1 | MS-0893 | W. M. Scherzinger, 9123 |
| 1 | MS-0893 | G. W. Wellman, 9123 |
| 1 | MS-1110 | D. E. Womble, 9214 |
| 1 | MS-1111 | K. D. Devine, 9226 |
| 1 | MS-1111 | C. T. Vaughan, 9224 |
| 1 | MS-1185 | J. Pott, 15417 |
| 1 | MS-9001 | J. L. Handrock, 1843 |
| 1 | MS-9042 | J. Crowell, 8727 |
| 1 | MS-9042 | J. Dike, 8727 |
| 1 | MS-9042 | R. Gilbert-O’Neil, 8727 |
| 1 | MS-9042 | P. M. Gullett, 8727 |
| 1 | MS-9042 | B. Kistler, 8727 |
| 1 | MS-9042 | C. Moen, 8728 |
| 1 | MS-9042 | V. Revelli, 8727 |
| 1 | MS-9042 | P. A. Spence, 8727 |
| 1 | MS-9161 | E-P Chen, 8726 |
| 1 | MS-9161 | P. A. Klein, 8726 |

| | | |
|---|---|---|
| 1 | MS-9018 | Central Technical Files, 8945-1 |
| 2 | MS-0899 | Technical Library, 9616 |
| 2 | MS-0612 | Review and Approval Desk, 9612, For DOE OSTI |