BISON Users Manual

BISON Release 1.3

J. D. Hales
K. A. Gamble
B. W. Spencer
S. R. Novascone
G. Pastore
W. Liu
D. S. Stafford
R. L. Williamson
D. M. Perez
R. J. Gardner
A. Casagranda
J. Galloway
C. Matthews
C. Unal
N. Carlson

Idaho National
Laboratory

BISON Users Manual

J. D. Hales
K. A. Gamble
B. W. Spencer
S. R. Novascone
G. Pastore
W. Liu
D. S. Stafford
R. L. Williamson
D. M. Perez
R. J. Gardner
A. Casagranda
J. Galloway
C. Matthews
C. Unal
N. Carlson

September 2016

Idaho National Laboratory
Fuel Modeling and Simulation Department
Idaho Falls, Idaho 83415

Contents

9

1 Introduction

BISON [1] is a finite element-based nuclear fuel performance code applicable to a variety of fuel forms including light water reactor fuel rods, TRISO particle fuel [2], and metallic rod [3] and plate fuel. It solves the fully-coupled equations of thermomechanics and species diffusion, for 1D spherically symmetric, 2D axisymmetric or 3D geometries. Fuel models are included to describe temperature and burnup dependent thermal properties, fission product swelling, densification, thermal and irradiation creep, fracture, and fission gas production and release. Plasticity, irradiation growth, and thermal and irradiation creep models are implemented for clad materials. Models are also available to simulate gap heat transfer, mechanical contact, and the evolution of the gap/plenum pressure with plenum volume, gas temperature, and fission gas addition. BISON is based on the MOOSE framework [4] and can therefore efficiently solve problems using standard workstations or very large high-performance computers.

Two input files are required as input when running BISON. One is a mesh file. While MOOSE supports several file formats, the ExodusII [5] format is the one used almost exclusively in BISON. This file commonly has "e" as its file extension. The mesh file may be generated using CUBIT [6] or another meshing tool. A further option is a meshing script bundled with BISON. This script, dependent on CUBIT and suitable for LWR fuel rod meshes, is the subject of Chapter 26.

The second file is a text file. This file commonly has "i" as its extension and contains a description of the variables, equations, boundary conditions, and material models associated with an analysis. The structure of the text input file is the main focus of this document.

2 Running BISON

2.1 Checking Out the Code

BISON uses GitLab for code management and distribution. Detailed instructions for checking out, building and contributing to the code can be found in the BISON Workshop slides located in the BISON repository (bison/docs/workshop) or https://bison.inl.gov.

2.2 Executing BISON

When first starting out with BISON, it is recommended to start from an example problem similar to the problem that you are trying to solve. Multiple examples can be found at bison/examples/ and bison/assessment/. It may be worth running the example problems to see how the code works and modifying input parameters to see how the run time, results and convergence behavior change.

To demonstrate running BISON, consider the `inputSmeared.i` example problem.

```
cd ~/projects/bison/examples/2D-RZ_rodlet_10pellets
# To run with one processor
~/projects/bison/bison-opt -i inputSmeared.i
# To run in parallel (4 processors)
mpiexec -n 4 ../../bison-opt -i inputSmeared.i
```

2.3 Getting Started

2.3.1 Input to BISON

Before running any problem, the power function, axial profile, mesh, and any functions needed for boundary conditions need to be generated.

Typically, a `PiecewiseLinear` function is used together with an external data file to specify a complex power history. This file has time and power specified in columns or rows, with the first row (or column) being the time (seconds) and the second row (or column) being power (W/m). Any data file that is used as input to BISON must be in Windows comma separated values (csv) format. Looking at `inputSmeared.i`, the power history is specified as:

```
[./power_history]
  type = PiecewiseLinear
  data_file = powerhistory.csv
  format = rows
  scale_factor = 1.0
```

```
[../]
```

The axial power profile, if present, is input as a `PiecewiseBilinearFile`. The axial peaking factors are input as a table within the file, with the top row being the axial location from the bottom of the rod and the left column as time. The axial peaking factors used for the example problem `inputSmeared.i` for the first three axial locations is as follows:

```
          9.44E-03, 1.54E-02, 2.13E-02
0.00E+00, 0.00E+00, 0.00E+00, 0.00E+00
1.00E+00, 5.37E-01, 8.68E-01, 1.01E+00
1.50E+08, 5.37E-01, 8.68E-01, 1.01E+00
```

The mesh can either be generated with the mesh script described in Chapter 26, or if you do not have CUBIT, you can generate a simple 2D-RZ axisymmetric mesh with smeared solid fuel pellets (single fuel column) with the `SmearedPelletMesh` within BISON. To generate the mesh similar to the one used in the example problem inputSmeared.i, the mesh block would look like:

```
[Mesh]
  type = SmearedPelletMesh
  clad_mesh_density = customize
  pellet_mesh_density = customize
  ny_p = 80 # Total number of axial elements in fuel
  nx_p = 11 # Number of radial elements in fuel
  nx_c = 5  # Number of elements through thickness of clad
  ny_cu = 3 # Number of axial element of upper clad gap
  ny_c = 80 # Number of axial elements of clad wall
  ny_cl = 3 # Number of axial elements of lower clad cap
  clad_thickness = 5.6e-4
  pellet_outer_radius = 0.0041
  clad_bot_gap_height = 1.0e-3
  pellet_quantity = 10
  pellet_height = 0.01186
  plenum_fuel_ratio = 0.045 # or use clad_top_gap_height = 3.0e-3
  clad_gap_width = 8e-5
  top_bot_clad_height = 2.24e-3
  elem_type = QUAD8
  displacements = 'disp_x disp_y'
  patch_size = 1000
[]
```

2.3.2 Post Processing

BISON typically writes solution data to an ExodusII file. Data may also be written in other formats, a simple comma separated file giving global data being the most common.

Several options exist for viewing ExodusII results files. These include commercial as well as open-source tools. One good choice is Paraview, which is open-source.

Paraview is available on a variety of platforms. It is capable of displaying node and element data in several ways. It will also produce line plots of global data or data from a particular node

or element. A complete description of Paraview is not possible here, but a quick overview of using Paraview with BISON results is available in the BISON workshop material.

2.3.3 Graphical User Interface

It is worth noting that a graphical user interface (GUI) exists for all MOOSE-based applications. This GUI is named Peacock. Information about Peacock and how to set it up for use may be found on the MOOSE wiki page.

Peacock may be used to generate a text input file. It is also capable of submitting the analysis. Finally, it provides basic post processing capabilities.

3 Overview

3.1 Basic Syntax

The input file used by BISON is broken into sections or blocks identified with square brackets. The type of input block is placed in the opening brackets, and empty brackets mark the end of the block.

```
[BlockName]
  <block lines and subblocks>
[]
```

Each block may have subblocks, which may in turn have subblocks. The `Functions` block, for example, will have multiple subblocks, each corresponding to a specific function. The line commands in the `Functions` subblocks will describe the function details.

subblocks are opened and closed as

```
  [./subblock_name]
    <line commands>
  [../]
```

Note that the name given in the subblocks must be unique when compared with all other subblocks in the current block.

Line commands are given as key/value pairs with an equal sign between them. They specify parameters to be used by the object being described. The key is a string (no whitespace), and the value may be a string, an integer, a real number, or a list of strings, integers, or real numbers. Lists are given in single quotes and are separated by whitespace.

Often subblocks will include a `type` line command. This line command specifies the particular type of object being described. The object type indicates which line commands are appropriate for describing the object. BISON will give an error message if a line command is given that does not apply for the current object type. An error message will also be given if a line command is repeated within the current block or if a line command is unused during the initial setup of the simulation.

In this document, line commands are shown with the keyword, an equal sign, and, in angle brackets, the value. If a default value exists for that line command, it is shown in parentheses.

In the initial description of a block, line commands common to all subblocks will be described. Those line commands are then omitted from the description of the subblocks but are nonetheless valid line commands for those subblocks.

The name of a subblock (`[./<name>]`) is most often arbitrary. However, the names of subblocks of `Variables`, `AuxVariables`, and `Postprocessors` define the names used for those entities.

3.2 BISON Syntax Page

A complete listing of all input syntax options in MOOSE is available on the MOOSE Documentation page. See the section on Input File Documentation. Note also that you can run `./bison-opt --dump` to get a list of valid input options for BISON.

3.3 Units

Because BISON uses several empirical models, BISON input expects SI units. This simplifies model input by eliminating the possibility of one set of units for one model and another set of units for a different model. Any needed unit conversions are done inside BISON.

3.4 High-Level Description of a BISON Simulation

The primary purpose of BISON is to solve coupled systems of partial differential equations (PDEs), where the equations represent important physics related to engineering scale nuclear fuel behavior. Fuel simulations typically consist of solving the following energy, momentum, and mass (or species) conservation equations,

$$\rho C_p \frac{\partial T}{\partial t} + \nabla \cdot \mathbf{q} - e_f \dot{F} = 0, \tag{3.1}$$

$$\nabla \cdot \sigma + \rho \mathbf{f} = 0. \tag{3.2}$$

$$\frac{\partial C}{\partial t} + \nabla \cdot \mathbf{J} + \lambda C - S = 0, \tag{3.3}$$

In Equation 3.1, T, ρ and C_p are the temperature, density and specific heat, respectively, e_f is the energy released in a single fission event, and $\dot{F}$ is the volumetric fission rate.

Momentum conservation (Equation 3.2) is prescribed assuming static equilibrium at each time increment where σ is the Cauchy stress tensor and $\mathbf{f}$ is the body force per unit mass (e.g. gravity). The displacement field u, which is the primary solution variable, is connected to the stress field via the strain, through a constitutive relation.

In the equation for species conservation (3.3) C, λ, and S are the concentration, radioactive decay constant, and source rate of a given species, respectively.

Often, fuels performance problems are limited to thermomechanics, where only Equations 3.1 and 3.2 are solved.

Each term in Equations 3.1 - 3.3 (time derivatives, divergence, source, sinks, etc.) are referred to as kernels and are discussed in greater detail in Chapter 14.

These equations are solved simultaneously using the finite element method (FEM) and JFNK approach [7] on a discretized domain. The domain (also referred to as a mesh) may represent uranium dioxide fuel pellets and zirconium clad in a light water reactor (LWR) simulation. Blocks, side sets, and node sets are defined on the mesh such that material models and boundary

conditions can be assigned to different parts of the model. Details regarding the mesh, material models, and boundary conditions can be found in chapters 6, 16, and 10 respectively.

Kernels, boundary conditions, and material models may require supporting information and calculations. This is achieved through the use of Functions and AuxKernels, which are detailed in chapters 9 and 12. For example, a function can be used to define power and time value pairs, which would inform the source term in the energy equation (Equation 3.1). An AuxKernel could be used to define fission rate or burnup, which could be used to inform material models that are dependent on those values. AuxKernels can also be used for writing information, such as stress components, to the output file.

Execution on the analysis is described in the Executioner block. Line commands describe time stepping details and solver options. See Chapter 19 for details.

MOOSE Postprocessors compute a single scalar value at each timestep. These can be minimums, maximums, averages, volumes, or any other scalar quantity. One example of the use of Postprocessors in BISON is computing the gas volume of an LWR rod. The gas volume changes timestep to timestep, but since it is a single scalar quantity, a Postprocessor computes this value. Chapter 18 gives examples.

The following sections delve deeper into the topics mentioned here. The format basically follows that of a typical BISON LWR input file and provides details for each section. Required parameters have **Required** included in their description throughout the document.

4 Global Parameters

The GlobalParams block specifies parameters that are available, as appropriate, in any other block or subblock in the input file. For example, imagine a subblock that accepts a line command with the keyword value. If the subblock has a line command for value, that line command will be used regardless of what is in GlobalParams. However, if the line command is missing in the subblock but defined in GlobalParams, the subblock will use the parameter defined in GlobalParams. In the example below, the line commands order = FIRST and family = LAGRANGE will be available in all blocks and subblocks in the remainder of the input file.

```
[GlobalParams]
  order = FIRST
  family = LAGRANGE
[]
```

5 Problem

The `Problem` block is typically only used to indicate that a model should run as axisymmetric (RZ) or spherically symmetric (RSPHERICAL). If the model is 3D, the `Problem` block may be omitted.

```
[Problem]
  coord_type = <string>
[]
```

There are two advanced cases that require a `[Problem]` block to be included in the input file. These cases are a known as `ReferenceResidualProblem` and `FrictionalContactProblem`. When using either of these types there are many required additions throughout the input file. Therefore `ReferenceResidualProblem` and `FrictionalContactProblem` are discussed in Chapters 24 and 25 respectively.

6 Mesh

The `Mesh` block's purpose is to give details about the finite element mesh to be used. Typically meshes for BISON simulations are created using the mesh generation tool Cubit (known as Trelis for non-DOE users). For simulations of LWR fuel there is a mesh script found in bison/tools/UO2/. The details of the mesh script are provided in Chapter 26.

```
[Mesh]
  file = <string>
  displacements = <string list>
  patch_size = <integer> (40)
[]
```

file	**Required**. This is the mesh file name. BISON uses ExodusII mesh files.
displacements	List of the displacement variables. This line must be given if the analysis is to use contact or nonlinear geometry. Typically 'disp_x disp_y' for an axisymmetric analysis.
patch_size	Number of nearby elements to consider as possible contacting surfaces. The value for the patch size depends upon whether Dirac or Constraint based contact is used. For Dirac a typical value is 1000. For Constraint it is ideal to choose a small enough patch size that encompasses all possible contacting surfaces to reduce memory requirements. For example, if the fuel moves up the clad 8 nodes make the patch size 20. This will allow the contact search to use 10 nodes above and 10 nodes belows the point at which the fuel comes into contact with the clad.

For users that do not have access to Cubit or Trelis but want to simulate LWR fuel there is a SmearedPelletMesh type that can be used to generate a mesh for modeling a smeared column of fuel (i.e. no dishes and or chamfers). The structure of the SmearedPelletMesh block is outlined below:

```
[Mesh]
  type = SmearedPelletMesh
  clad_mesh_density = <string> (medium)
  pellet_mesh_density = <string> (medium)
  ny_p = <integer> (24)
  nx_p = <integer> (8)
  nx_c = <integer> (2)
  ny_cu = <integer> (1)
  ny_c = <integer> (24)
  ny_cl = <integer> (1)
```

```
   bx_p = <real> (1)
   clad_thickness = <real> (0.00041)
   pellet_outer_radius = <real> (0.0041)
   clad_bot_gap_height = <real> (0.00127)
   pellet_quantity = <real> (2)
   elem_type = <string> (QUAD4)
   displacements = <string list>
   patch_size = <integer> (4)
[]
```

type	SmearedPelletMesh
clad_mesh_density	Mesh density of the clad. Choices are coarse, medium, fine or custom. Default is medium.
pellet_mesh_density	Mesh density of the fuel pellets. Choices are coarse, medium, fine or custom. Default is medium.
ny_p	Number of finite elements in a fuel pellet in the axial direction.
nx_p	Number of finite elements in a fuel pellet in the radial direction.
nx_c	Number of finite elements through the thickness of the cladding in the radial direction.
ny_cu	Number of finite elements through the thickness of the cladding in the axial direction of the upper plug.
ny_c	Number of finite elements axially through the cladding.
ny_cl	Number of finite elements through the thickness of the cladding in the axial direction of the lower plug.
bx_p	The amount to grow (or shrink) the fuel elements in the radial direction. Range is 0.5 to 2.0.
clad_thickness	The cladding thickness.
pellet_outer_radius	The outer radius of the pellet.
clad_bot_gap_height	Gap between bottom of pellet stack and the inside bottom surface of the cladding.
pellet_quantity	Number of pellets to be included.
pellet_height	The height of the pellet.
plenum_fuel_ratio	Ratio of the axial gas height to the fuel height inside the cladding. Either plenum_fuel_ratio or clad_top_gap_height must be specified but not both.
clad_top_gap_height	Gap between top of pellet and inside top surface of cladding. Either plenum_fuel_ratio or clad_top_gap_height must be specified but not both.
clad_gap_width	Gap between outer radius of pellet and inside surface of cladding.
top_bot_clad_height	Thickness of top and bottom cladding walls.

`elem_type`	Type of finite element. Default is QUAD4. For second-order meshes use QUAD8.
`displacements`	List of the displacement variables. This line must be given if the analysis is to use contact or nonlinear geometry. Typically ′`disp_x disp_y`′ for an axisymmetric analysis.
`patch_size`	Number of nearby elements to consider as possible contacting surfaces. The value for the patch size depends upon whether Dirac or Constraint based contact is used. For Dirac a typical value is 1000. For Constraint it is ideal to choose a small enough patch size that encompasses all possible contacting surfaces to reduce memory requirements. For example, if the fuel moves up the clad 8 nodes make the patch size 20. This will allow the contact search to use 10 nodes above and 10 nodes belows the point at which the fuel comes into contact with the clad.

7 Variables

The Variables block is where all of the primary solution variables are identified. The name of each variable is taken as the name of the subblocks. Primary solution variables often include temperature (usually named temp) and displacement (usually named disp_x, disp_y, and disp_z).

```
[Variables]
  [./var1]
    order = <string>
    family = <string>
  [../]
  [./var2]
    order = <string>
    family = <string>
    initial_condition = <real>
    scaling = <real> (1)
  [../]
[]
```

order	The order of the variable. Typical values are FIRST and SECOND.
family	The finite element shape function family. A typical value is LAGRANGE.
initial_condition	Optional initial value to be assigned to the variable. Zero is assigned if this line is not present.
scaling	Amount to scale the variable during the solution process. This scaling affects only the residual and preconditioning steps and not the final solution values. This line command is sometimes helpful when solving coupled systems where one variable's residual is orders of magnitude different that the other variables' residuals.

8 AuxVariables

The AuxVariables block is where all of the auxiliary variables are identified. The name of each variable is taken as the name of the subblocks. Auxiliary variables are used for quantities such as fast neutron flux, element-averaged stresses, and other output variables.

```
[AuxVariables]
  [./var1]
    order = <string>
    family = <string>
  [../]
  [./var2]
    order = <string>
    family = <string>
    initial_condition = <real>
  [../]
[]
```

order	The order of the variable. Typical values are CONSTANT, FIRST, and SECOND.
family	The finite element shape function family. Typical values are MONOMIAL and LAGRANGE.
initial_condition	Optional initial value to be assigned to the variable. Zero is assigned if this line is not present.

9 Functions

9.1 Composite

The `Composite` function takes an arbitrary set of functions, provided in the `functions` parameter, evaluates each of them at the appropriate time and position, and multiplies them together. The function can optionally be multiplied by a scale factor, which specified using the `scale_factor` parameter.

```
[./composite]
  type = CompositeFunction
  functions = <string list>
  scale_factor = <real> (1.0)
[../]
```

type	CompositeFunction
functions	List of functions to be multiplied together.
scale_factor	Scale factor to be applied to resulting function. Default is 1.

9.2 ParsedFunction

The `ParsedFunction` function takes a mathematical expression in `value`. The expression can be a function of time (t) or coordinate (x, y, or z). The expression can include common mathematical functions. Examples include '4e4+1e2*t', 'sqrt(x*x+y*y+z*z)', and 'if(t<=1.0, 0.1*t, (1.0+0.1)*cos(pi/2*(t-1.0)) - 1.0)'. Constant variables may be used in the expression if they have been declared with `vars` and defined with `vals`. Further information can be found at http://warp.povusers.org/FunctionParser/.

```
[./parsedfunction]
  type = ParsedFunction
  value = <string>
  vals = <real list>
  vars = <string list>
[../]
```

type	ParsedFunction
value	**Required**. String describing the function.
vals	Values to be associated with variables in `vars`.
vars	Variable names to be associated with values in `vals`.

9.3 PiecewiseBilinear

The `PiecewiseBilinear` function reads a csv file and interpolates values based on the data in the file. The interpolation is based on x-y pairs. If `axis` is given, time is used as the y index. Either `xaxis` or `yaxis` or both may be given. Time is used as the other index if one of them is not given. If `radius` is given, `xaxis` and `yaxis` are used to orient a cylindrical coordinate system, and the x-y pair used in the query will be the radial coordinate and time.

```
[./piecewiselinear]
  type = PiecewiseBilinear
  data_file = <string>
  axis = <0, 1, or 2 for x, y, or z>
  xaxis = <0, 1, or 2 for x, y, or z>
  yaxis = <0, 1, or 2 for x, y, or z>
  scale_factor = <real> (1.0)
  radial = <bool> (false)
[../]
```

type	PiecewiseBilinear
data_file	File holding your csv data.
axis	Coordinate direction to use in the function evaluation.
xaxis	Coordinate direction used for x-axis data.
yaxis	Coordinate direction used for y-axis data.
scale_factor	Scale factor to be applied to resulting function. Default is 1.
radial	Set to `true` if interpolation should be done along a radius rather than along a specific axis. Requires `xaxis` and `yaxis`.

9.4 PiecewiseConstant

The `PiecewiseConstant` function defines the data using a set of x-y data pairs. Instead of linearly interpolating between the values, however, the `PiecewiseConstant` function is constant when the abscissa is between the values provided by the user. The `direction` parameter controls whether the function takes the value of the abscissa of the user-provided point to the right or left of the value at which the function is evaluated.

```
[./piecewiseconstant]
  type = PiecewiseConstant
  x = <real list>
  y = <real list>
  xy_data = <real list>
  data_file = <string>
  format = <string> (rows)
  scale_factor = <real> (1.0)
  axis = <0, 1, or 2 for x, y, or z>
  directon = <string> (left)
```

```
[../]
```

type	PiecewiseConstant
x	List of x values for x-y data.
y	List of y values for x-y data.
xy_data	List of pairs of x-y data points.
data_file	Name of an file containing x-y data.
format	Format of x-y data in external file.
scale_factor	Scale factor to be applied to resulting function. Default is 1.
axis	Coordinate direction to use in the function evaluation. If not present, time is used as the function input.

9.5 PiecewiseLinear

The PiecewiseLinear function performs linear interpolations between user-provied pairs of x-y data. The x-y data can be provided in three ways. The first way is through a combination of the x and y paramaters, which are lists of the x and y coordinates of the data points that make up the function. The second way is in the xy_data parameter, which is a list of pairs of x-y data that make up the points of the function. This allows for the function data to be specified in columns by inserting line breaks after each x-y data point. Finally, the x-y data can be provided in an external file containing comma-separated values. The file name is provided in data_file, and the data can be provided in either rows (default) or columns, as specified in the format parameter.

By default, the x-data corresponds to time, but this can be changed to correspond to x, y, or z coordinate with the axis line. If the function is queried outside of its range of x data, it returns the y value associated with the closest x data point.

```
[./piecewiselinear]
  type = PiecewiseLinear
  x = <real list>
  y = <real list>
  xy_data = <real list>
  data_file = <string> (rows)
  format = <string>
  scale_factor = <real> (1.0)
  axis = <0, 1, or 2 for x, y, or z>
[../]
```

type	PiecewiseLinear
x	List of x values for x-y data.
y	List of y values for x-y data.
xy_data	List of pairs of x-y data points.

`data_file`	Name of an file containing x-y data.
`format`	Format of x-y data in external file.
`scale_factor`	Scale factor to be applied to resulting function. Default is 1.
`axis`	Coordinate direction to use in the function evaluation. If not present, time is used as the function input.

10 Boundary Conditions

The `BCs` block is for specifying various types of boundary conditions.

```
[BCs]
  [./name]
    type = <BC type>
    boundary = <string list>
    ...
  [../]
[]
```

type   Type of boundary condition.

boundary   List of boundaries (side sets). Either boundary numbers or names.

10.1 BulkCoolantBC

The `BulkCoolantBC` boundary condition determines the heat transfer from a boundary based upon a bulk coolant temperature and coolant heat tansfer coefficient.

```
  [./bulkcoolantBC]
    type = BulkCoolantBC
    variable = <variable>
    boundary = <string list>
    bulk_temperature = <real> (800)
    heat_transfer_coefficient = <real> (2000)
  [../]
```

type	BulkCoolantBC
variable	**Required**. Primary variable associated with this boundary condition.
boundary	**Required**. List of boundary names or ids where this boundary condition will apply.
bulk_temperature	The bulk coolant temperature.
heat_transfer_coefficient	The heat transfer coefficient of the coolant.

10.2 ConvectiveFluxBC

The `ConvectiveFluxBC` boundary condition determines the value on a boundary based upon the initial and final values, the flux through the boundary and the duration of exposure..

```
[./convectivefluxBC]
  type = ConvectiveFluxBC
  variable = <variable>
  boundary = <string list>
  initial = <real> (500)
  final = <real> (500)
  rate = <real> (7500)
[../]
```

type	ConvectiveFluxBC
variable	**Required**. Primary variable associated with this boundary condition.
boundary	**Required**. List of boundary names or ids where this boundary condition will apply.
initial	The initial value of the variable on the boundary.
final	The final value of the variable on the boundary.
rate	The flux of the variable through the boundary.

10.3 ConvectiveFluxFunction

The `ConvectiveFluxFunction` boundary condition determines the value on a boundary based upon the heat transfer coefficient of the fluid on the outside of boundary and far-field temperature.

```
[./convectivefluxFunction]
  type = ConvectiveFluxFunction
  variable = <variable>
  boundary = <string list>
  T_infinity= <string>
  coefficient = <real>
  coefficient_function = <string>
[../]
```

type	ConvectiveFluxFunction
variable	**Required**. Primary variable associated with this boundary condition.
boundary	**Required**. List of boundary names or ids where this boundary condition will apply.
T_infinity	**Required**. The name of the function describing the far-field temperature.

coefficient	**Required**. The heat transfer coefficient of the fluid in contact with the boundary. If coefficient_function is provided this coefficient multiplies the function.
coefficient_function	Function describing the heat transfer coefficient.

10.4 CoolantChannel

The effect of the coolant on the heat transfer at the exterior cladding surface can be modeled using the CoolantChannel feature. This feature appears in the input file in its own block (i.e., not inside the BCs block).

The presence of some input parameters causes others to be ignored. The following describes the input parameter precedence.

If heat_transfer_coefficient is given, its value will be assigned to the given boundary. All other parameters related to the heat transfer coefficient calculation are ignored.

Enthalpy is taken as coupledEnthalpy if present. Otherwise, heat flux is calculated based on linear_heat_rate, specification of number_axial_zone, and specification of heat_flux, in highest precedence order. The integrated heat flux is computed based on the same precedence. As an example, if number_axial_zone and heat_flux are specified, heat_flux will be ignored. These are used as inputs to the heat transfer coefficient correlations.

The coolant_material is water by default. It can be instead set to sodium, and a heat transfer correlation for liquid sodium is used. With sodium coolant, calculations for phase change are disabled. By default, the subchannel geometry is set to triangular if the user selects sodium coolant, but this can be changed to a square channel if desired.

```
[CoolantChannel]
  [./coolantchannel]
    boundary = <string list>
    variable = <string>
    axial_power_profile = <string>
    blockage_ratio = <real> (0)
    chf_correlation_type=<int> (4)
    compute_enthalpy =<bool> (true)
    cond_metal = <real>
    cond_oxide = <real>
    coolant_material = <string>
    coupledEnthalpy = <string>
    direction = <string>
    direction2 = <string>
    flooding_rate = <real>
    flooding_time = <real>
    flow_area = <real>
    fuel_stack_length = <real>
    fuel_stack_bottom = <real>
    heat_flux = <string>
    heat_transfer_coefficient = <string or real>
    heat_transfer_mode = <string> (0)
```

```
    heated_diameter = <real>
    heated_perimeter = <real>
    htc_correlation_type = <string>
    hydraulic_diameter = <real>
    initial_power = <real>
    initial_temperature = <real>
    inlet_massflux = <string or real>
    inlet_pressure = <string or real>
    inlet_temperature = <string or real>
    input_Tchf = <real> (0)
    input_Tmin = <real> (0)
    input_rewetting_htc = <real> (1.0e5)
    linear_heat_rate = <string>
    model_post_chf = <bool> (true)
    number_axial_zone = <integer> (0)
    number_lateral_zone = <integer> (1)
    oxide_thickness = <string>
    oxide_model = <string> (zirconia)
    pbr = <real>
    reflooding_model = <int> (1)
    rod_diameter = <real> (0.01)
    rod_pitch = <real> (0.0126)
    specified_height = <real> (0)
    subchannel_geometry = <string>
  [../]
[]
```

boundary	**Required**. List of boundaries. Typically only one boundary id is given.
variable	**Required**. Name of variable associated with this BC. Typically temp.
axial_power_profile	Function name for function describing axial power factors.
blockage_ratio	Flow blockage ratio used in FLECHT correlations.
chf_correlation_type	CHF correlatons. one of 1 for EPRI, 2 for GE, 3 for Zuber, and 4 for BIASI.
compute_enthalpy	option to turn on /off the enthalpy calculation.
cond_metal	Conductivity of the metal. Used if oxide_model is user.
cond_oxide	Conductivity of the oxide. Used if oxide_model is user.
coolant_material	Water or sodium. Defaults to water.
coupledEnthalpy	Variable name. If given, enthalpy is taken from this variable directly instead of being calculated.
direction	One of x, y, or z. Coordinate direction associated with fluid flow. Default is y.

direction2	One of x, y, or z. Coordinate direction associated with lateral dimension of model. Default is x. This input is used for plate geometry.
flooding_rate	Inlet fooding rate.
flooding_time	The starting time of flooding.
flow_area	Flow area. If used, must be used with `heated_diameter`, `heated_perimeter`, and `hydraulic_diameter`. If used, `rod_diameter` and `rod_pitch` will be ignored.
fuel_stack_bottom	The axial position of the fuel stack bottom.
fuel_stack_length	The length of fuel stacks.
heat_flux	Function name for function describing the heat flux at the cladding surface.
heat_transfer_coefficient	Either a function name for a function describing the heat transfer coefficient or a real value to be assigned as the heat transfer coefficient. If present, other parameters controlling the heat transfer coefficient calculation will be ignored.
heat_transfer_mode	One of 0 (automatic), 1 (natural convection), 2 (forced liquid convection), 3 (subcooled boiling), 4 (saturated boiling), 5 (transition boiling), 6 (film boiling), and 7 (single phase vapor).
heated_diameter	Heated diameter. If used, must be used with `flow_area`, `heated_perimeter`, and `hydraulic_diameter`. If used, `rod_diameter` and `rod_pitch` will be ignored.
heated_perimeter	Heated perimeter. If used, must be used with `flow_area`, `heated_diameter`, and `hydraulic_diameter`. If used, `rod_diameter` and `rod_pitch` will be ignored.
htc_correlation_type	One of 1 (Thom), 2 (Jens Lottes), 3(Chen) or 4 (Shrock-Grossman) for pre-CHF correlations;
	or 1 (McDonough-Milich-King) and 2 (modified Condie-Bengtson) for transition boiling correlations;
	or 1 (Groenveld) and 2 (Dougall-Rohsenow) for film boiling correlations.
hydraulic_diameter	Hydraulic diameter. If used, must be used with `flow_area`, `heated_perimeter`, and `heated_diameter`. If used, `rod_diameter` and `rod_pitch` will be ignored.
initial_power	Initial peak power (kW/m).
initial_temperature	Initial peak clad temperature.
inlet_massflux	Either a function name for a function describing the inlet mass flux or a real value to be assigned as the inlet mass flux.

inlet_pressure	Either a function name for a function describing the inlet pressure or a real value to be assigned as the inlet pressure.
inlet_temperature	Either a function name for a function describing the inlet temperature or a real value to be assigned as the inlet temperature.
input_Tchf	Input temperature at critical heat flux.
input_Tmin	Input rewetting temperature.
input_rewetting_htc	Input rewetting heat transfer coefficient.
linear_heat_rate	Function name for a function describing the linear heat rate.
number_axial_zone	Number of axial divisions along the cladding to be used in integrating the heat flux.
number_lateral_zone	Number of lateral divisions along the cladding to be used in integrating the heat flux. This input is used for plate geometry.
model_post_chf	Option to turn on or off post-CHF calculations.
oxide_thickness	Name of AuxVariable representing the oxide thickness. If not given, the calculated heat transfer coefficient will not account for an oxide layer.
oxide_model	One of zirconia, alumina, or user.
reflooding_model	Model options for modeling reflooding : either 0 *oldcorrelation* or 1 *newWestinghousecorrelation* .
rod_diameter	Diameter of the fuel rod.
rod_pitch	Pitch or spacing between fuel rods.
specified_height	The input to compute reflooding heat transfer at a specified axial location; used for testing purpose only.
subchannel_geometry	Geometry of the pin array: either square or triangular. If not specified, the geometry is square if the coolant is water or triangular if the coolant is sodium.

10.5 Dirichlet

10.5.1 DirichletBC

```
[./dirichletbc]
  type = DirichletBC
  variable = <variable>
  boundary = <string list>
  value = <real>
[../]
```

type	DirichletBC
variable	**Required**. Primary variable associated with this boundary condition.
boundary	**Required**. List of boundary names or ids where this boundary condition will apply.
value	**Required**. Value to be assigned.

10.5.2 PresetBC

The PresetBC takes the same inputs as DirichletBC and also acts as a Dirichlet boundary condition. However, the implementation is slightly different. PresetBC causes the value of the boundary condition to be applied before the solve begins where DirichletBC enforces the boundary condition as the solve progresses. In certain situations, one is better than another.

10.5.3 FunctionDirichletBC

```
[./functiondirichletbc]
  type = FunctionDirichletBC
  variable = <variable>
  boundary = <string list>
  function = <string>
[../]
```

type	FunctionDirichletBC
variable	**Required**. Primary variable associated with this boundary condition.
boundary	**Required**. List of boundary names or ids where this boundary condition will apply.
function	**Required**. Function that will give the value to be applied by this boundary condition.

10.5.4 FunctionPresetBC

The FunctionPresetBC takes the same inputs as FunctionDirichletBC and also acts as a Dirichlet boundary condition. However, the implementation is slightly different. FunctionPresetBC causes the value of the boundary condition to be applied before the solve begins where FunctionDirichletBC enforces the boundary condition as the solve progresses. In certain situations, one is better than another.

10.6 DryCaskHeatFlux

The DryCaskHeatFlux BC is used to model the heat flux from a rod in the center of a fuel assembly, stored inside a dry cask storage system (DCSS). The flux includes radiative and conductive effects inside the assembly and conductive/convective effects from the assembly to ambient.

```
[./decay_heat_in_dcss]
  type = DryCaskHeatFlux
  variable = <variable>
  boundary = <string list>
  bwr_or_pwr = <string>
  fill_gas = <string>
  ambient_temperature = <real>
  cask_effective_htc = <real>
  start_time = <real>
  drying_duration = <real>
[../]
```

type	DryCaskHeatFlux
variable	**Required**. Primary variable associated with this boundary condition.
boundary	**Required**. List of boundaries where this BC will apply.
bwr_or_pwr	**Required**. Whether to use a typical geometry from a BWR or a PWR assembly.
fill_gas	**Required**. helium, nitrogen, or vacuum.
ambient_temperature	Temperature outside the cask.
cask_effective_htc	**Required**. Effective heat transfer coefficient from assembly to ambient (W/K).
start_time	The time when this BC will begin.
drying_duration	If drying (vacuum) is desired, it will be applied immediately after the start_time. After drying in vacuum, the calculation switches from vacuum fill gas to fill_gas specified by the user.

10.7 HydrogenPickup

The HydrogenPickup BC is used to model the flux of hydrogen into the clad that is caused by oxide growth. The flux is approximated as a constant fraction of the hydrogen liberated by oxide growth at the interface between the coolant water and the clad.

Note that this BC must be coupled to a variable that gives the thickness of the oxide over time, such as with the OxideAux kernel. For this to work properly, OxideAux must be set to update on updates to the residual; *it will not work if the OxideAux is set to update on time steps*.

```
[./hydrogen_pickup]
  type = HydrogenPickup
  variable = <variable>
  boundary = <string list>
  oxide_thickness = <variable>
  pickup_fraction = <real> (0.15)
  clad_thickness = <real> (660e-6)
```

35

```
    fuel_pin_geometry = <string>
  [../]
```

type	HydrogenPickup
variable	**Required**. Primary variable associated with this boundary condition.
boundary	**Required**. List of boundary names or ids where this boundary condition will apply.
oxide_thickness	**Required**. The coupled variable that gives the oxide thickness on the boundary.
pickup_fraction	The fractional amount of hydrogen liberated by the oxide growth that is absorbed into the clad.
clad_thickness	The initial thickness of the clad; only needed if fuel_pin_geometry is not specified.
fuel_pin_geometry	Name of the FuelPinGeometry object (see 23.1).

10.8 PlenumPressure

The PlenumPressure block is used to specify internal rod pressure as a function of temperature, cavity volume, and moles of gas.

The PlenumPressure boundary condition uses two levels of nesting within the BCs block. This allows the pressure to be applied properly in all coordinate directions although it is specified one time only.

The volume and pressure specified in the plenum pressure block along with the initial condition specified in the temperature variable block are used to calculate the initial moles. The initial moles are then used to update the plenum pressure throughout the simulation. It is worth noting to make sure the initial temperature is set to the temperature of the gas when fabricated, usually room temperature (293 K).

The postprocessors coupled to the plenum pressure boundary condition (gas volume and rod interior temperature) need to be executed at each residual such that the plenum pressure is calculated for that specific timestep. If calculated at each timestep, the calculation uses volume and temperature from the previous step to calculate the plenum pressure for the current step, causing a lag in the plenum pressure used and reported for that timestep.

```
[./PlenumPressure]
  [./plenumpressure]
    boundary = <string list>
    initial_pressure = <real> (0)
    initial_temperature = <real>
    startup_time = <real> (0)
    R = <real>
    output_initial_moles = <string>
    temperature = <string>
    volume = <string>
```

```
    material_input = <string list>
    output = <string>
    refab_time = <real list>
    refab_pressure = <real list>
    refab_volume = <real list>
    refab_type = <integer list>
  [../]
[../]
```

boundary	**Required**. List of boundary names or ids where this boundary condition will apply.
initial_pressure	The initial pressure in the plenum.
initial_temperature	The initial temperature of the plenum. If not given, will use the initial value from the Postprocessor given by temperature.
startup_time	The amount of time during which the pressure will ramp from zero to its true value.
R	**Required**. The universal gas constant. In BISON, SI units are used, and R should be 8.3143.
output_initial_moles	If given, the name to use to report the initial moles of gas.
temperature	**Required**. The name of the Postprocessor holding the average temperature value.
volume	**Required**. The name of the Postprocessor holding the internal volume.
material_input	The name of the Postprocessors that hold the amount of material injected into the plenum.
output	If given, the name to use for reporting the plenum pressure value. If not given, the block name will be used.
refab_time	The time(s) at which the plenum pressure must be reinitialized (likely due to fuel rod refabrication).
refab_pressure	The pressure of fill gas at refabrication. Number of values must match number in refab_time.
refab_temperature	The temperature at refabrication. Number of values must match number in refab_time.
refab_volume	The gas volume at refabrication. Number of values must match number in refab_time.

10.9 Pressure

The Pressure boundary condition uses two levels of nesting within the BCs block. This allows the pressure to be applied properly in all coordinate directions although it is specified one time only.

```
[./Pressure]
  [./pressure]
    boundary = <string list>
    factor = <real> (1)
    function = <string>
  [../]
[../]
```

boundary   **Required**. List of boundary names or ids where this boundary condition will apply.

factor   Magnitude of pressure to be applied. If `function` is also given, `factor` is multiplied by the output of the function and then applied as the pressure.

function   Function that will give the value to be applied by this boundary condition.

11 Contact

Finite element contact enforces constraints between surfaces in the mesh. Mechanical contact prevents penetration and develops contact forces. Thermal contact transfers heat between the surfaces. In BISON there are currently two systems to choose from for mechanical contact: Dirac and Constraint. Constaint based contact is recommended for two-dimensional problems and Dirac for three-dimensional problems. Constraint contact is more robust but due to the patch size requirement specified in the `Mesh` block constraint contact uses too much memory on 3D problems. Depending upon the contact formalism chosen the solver options to be used change. The details of the solver parameters recommended for Dirac and Constraint contact formalisms are provided in Section 19.2.

11.1 Mechanical Contact

```
[Contact]
  [./contact]
    disp_x = <variable>
    disp_y = <variable>
    disp_z = <variable>
    formulation = <string> (DEFAULT)
    friction_coefficient = <real> (0)
    master = <string>
    model = <string> (frictionless)
    normal_smoothing_distance = <real>
    normal_smoothing_method = <string> (edge_based)
    order = <string> (FIRST)
    penalty = <real> (1e8)
    normalize_penalty = <bool> (false)
    slave = <string>
    system = <string> (Dirac)
    tangential_tolerance = <real>
    tension_release = <real> (0)
  [../]
[]
```

disp_x	**Required**. Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.

disp_z	Variable name for displacement variable in z direction. Typically `disp_z`.
formulation	One of DEFAULT, KINEMATIC, or PENALTY. DEFAULT is KINEMATIC.
friction_coefficient	The friction coefficient.
master	**Required**. The boundary id for the master surface.
model	One of frictionless, glued, or coulomb.
normal_smoothing_distance	Distance from face edge in parametric coordinates over which to smooth the contact normal. 0.1 is a reasonable value.
normal_smoothing_method	One of edge_based or nodal_normal_based. If nodal_normal_based, must also have a `NodalNormals` block.
order	The order of the variable. Typical values are `FIRST` and `SECOND`.
penalty	The penalty stiffness value to be used in the constraint.
normalize_penalty	Whether to normalize the penalty stiffness by the nodal area of the slave node.
slave	**Required**. The boundary id for the slave surface.
system	The system to use for constraint enforcement. Options are Dirac (DiracKernel) or Constraint. The default system is Dirac.
tangential_tolerance	Tangential distance to extend edges of contact surfaces.
tension_release	Tension release threshold. A node will not be released if its tensile load is below this value. If negative, no tension release will occur.

In LWR fuel analysis, the cladding surface is typically the master surface, and the fuel surface is the slave surface. It is good practice to make the master surface the coarser of the two.

The robustness and accuracy of the mechanical contact algorithm is strongly dependent on the penalty parameter. If the parameter is too small, inaccurate solutions are more likely. If the parameter is too large, the solver may struggle.

The DEFAULT option uses an enforcement algorithm that moves the internal forces at a slave node to the master face. The distance between the slave node and the master face is penalized. The PENALTY algorithm is the traditional penalty enforcement technique.

11.2 Thermal Contact

11.2.1 GapHeatTransfer

```
[ThermalContact]
  [./thermalcontact]
```

```
    type = GapHeatTransfer
    cylinder_axis_point_1 = <RealVectorValue>
    cylinder_axis_point_2 = <RealVectorValue>
    disp_x = <variable>
    disp_y = <variable>
    disp_z = <variable>
    emissivity_1 = <real> (0)
    emissivity_2 = <real> (0)
    gap_conductivity = <real> (1)
    gap_conductivity_function = <string>
    gap_conductivity_function_variable = <string>
    gap_geometry_type = <MooseEnum>
    master = <string>
    min_gap = <real> (1e-6)
    max_gap = <real> (1e6)
    normal_smoothing_distance = <real>
    normal_smoothing_method = <string> (edge_based)
    order = <string> (FIRST)
    quadrature = <bool> (false)
    slave = <string>
    sphere_origin = <RealVectorValue>
    stefan_boltzmann = <real> (5.669e-8)
    tangential_tolerance = <real>
    variable = <string>
  [../]
[]
```

type	GapHeatTransfer
cylinder_axis_point_1	Start point for defining cylindrical axis
cylinder_axis_point_2	End point for defining cylindrical axis
disp_x	Variable name for displacement variable in x direction. Typically disp_x. Optional.
disp_y	Variable name for displacement variable in y direction. Typically disp_y. Optional.
disp_z	Variable name for displacement variable in z direction. Typically disp_z. Optional.
emissivity_1	The emissivity of the fuel surface.
emissivity_2	The emissivity of the cladding surface.
gap_conductivity	The thermal conductivity of the gap material.
gap_conductivity_function	Thermal conductivity of the gap material as a function. Multiplied by gap_conductivity.
gap_conductivity_function_variable	Variable to be used in thermal_conductivity_function in place of time.

gap_geometry_type	Gap calculation type. Choices are: PLATE CYLINDER SPHERE
master	**Required**. The boundary id for the master surface.
min_gap	The minimum permissible gap size.
max_gap	The maximum permissible gap size.
normal_smoothing_distance	Distance from face edge in parametric coordinates over which to smooth the contact normal. 0.1 is a reasonable value.
normal_smoothing_method	One of edge_based or nodal_normal_based. If nodal_normal_based, must also have a NodalNormals block.
order	The order of the variable. Typical values are FIRST and SECOND.
quadrature	Whether or not to use quadrature point-based gap heat transfer.
slave	**Required**. The boundary id for the slave surface.
sphere_origin	Origin for sphere geometry
stefan_boltzmann	The Stefan-Boltzmann constant.
tangential_tolerance	Tangential distance to extend edges of contact surfaces.
variable	**Required**. The temperature variable name.

The quadrature option is recommended with second-order meshes. Also note that the type of conductance used depends on the value of the gap_geometry_type parameter (PLATE, CYLINDER, or SPHERE). An example of how to use this can be found in moose/modules/combined/tests/gap_heat_transfer_htonly/.

11.2.2 GapHeatTransferLWR

GapHeatTransferLWR differs from GapHeatTransfer in that the gap conductivity is computed based on the gases in the gap. To this may also be added the effect of solid-solid conduction. The gas in the gap may be flushed in a refabrication step. (See also PlenumPressure (10.8).) Also note that parameters gap_geometry_type, cylinder_axis_1 and 2, and sphere_origin can be used in the this block.

```
[ThermalContact]
  [./thermalcontact]
    type = GapHeatTransferLWR
    contact_coef = <real> (10)
    contact_pressure = <string>
    disp_x = <variable>
    disp_y = <variable>
```

```
    disp_z = <variable>
    emissivity_1 = <real> (0)
    emissivity_2 = <real> (0)
    external_pressure = <real> (0)
    initial_gas_fractions = <real list> (1 0 0 0 0 0 0 0 0 0)
    initial_moles = <string>
    gas_released = <string list>
    gas_released_fractions = <real list> (0 0 0.153 0.847 0 0 0 0 0 0)
    jump_distance_fuel = <real> (0)
    jump_distance_clad = <real> (0)
    jump_distance_model = <string> (DIRECT)
    master = <string>
    meyer_hardness <real> (0.68e9)
    min_gap = <real> (1e-6)
    max_gap = <real> (1e6)
    normal_smoothing_distance = <real>
    normal_smoothing_method = <string> (edge_based)
    order = <string> (FIRST)
    quadrature = <bool> (false)
    refab_gas_fractions = <real list>
    refab_time = <real list>
    refab_type = <integer list>
    roughness_fuel = <real> (1e-6)
    roughness_clad = <real> (1e-6)
    roughness_coef = <real> (1.5)
    interaction_layer = <integer> (0)
    slave = <string>
    stefan_boltzmann = <real> (5.669e-8)
    tangential_tolerance = <real>
    variable = <string>
  [../]
[]
```

type	GapHeatTransferLWR
contact_coef	The leading coefficient on the solid-solid conduction relation ($1/\sqrt{m}$).
contact_pressure	The contact pressure variable. Typically contact_pressure.
disp_x	Variable name for displacement variable in x direction. Typically disp_x. Optional.
disp_y	Variable name for displacement variable in y direction. Typically disp_y. Optional.
disp_z	Variable name for displacement variable in z direction. Typically disp_z. Optional.
emissivity_1	The emissivity of the fuel surface.

`emissivity_2`	The emissivity of the cladding surface.
`external_pressure`	The external (gas) pressure.
`initial_gas_fractions`	The initial fractions of constituent gases (helium, argon, krypton, xenon, hydrogen, nitrogen, oxygen, carbon monoxide, carbon dioxide, water vapor).
`initial_moles`	The `Postprocessor` that will give the initial moles of gas.
`gas_released`	List of one or more `Postprocessor`s that give the gas released.
`gas_released_fractions`	The fraction of released gas that is assigned to helium, argon, krypton, xenon, hydrogen, nitrogen, oxygen, carbon monoxide, carbon dioxide, and water vapor. One set of fractions for each `Postprocessor` listed in `gas_released`.
`jump_distance_fuel`	The temperature jump distance of the fuel.
`jump_distance_clad`	The temperature jump distance of the clad.
`jump_distance_model`	One of DIRECT (specify distances directly) or KENNARD (jump distances computed based on gas properties).
`master`	The boundary id for the master surface.
`meyer_hardness`	The Meyer hardness of the softer material (Pa).
`min_gap`	The minimum permissible gap size.
`max_gap`	The maximum permissible gap size.
`normal_smoothing_distance`	Distance from face edge in parametric coordinates over which to smooth the contact normal. 0.1 is a reasonable value.
`normal_smoothing_method`	One of edge_based or nodal_normal_based. If nodal_normal_based, must also have a `NodalNormals` block.
`order`	The order of the variable. Typical values are `FIRST` and `SECOND`.
`plenum_pressure`	The name of the plenum pressure `Postprocessor`.
`quadrature`	Whether or not to use quadrature point-based gap heat transfer.
`refab_gas_fractions`	The fractions of constituent gases at refabrication (helium, argon, krypton, xenon, hydrogen, nitrogen, oxygen, carbon monoxide, carbon dioxide, water vapor).
`refab_time`	The time(s) at which refabrication occurs. If multiple times are given, multiple sets of `refab_gas_fractions` and multiple `refab_types` must be given.
`refab_type`	One of 0 (instantaneous reset, evolving gas fraction thereafter) or 1 (instantaneous reset, constant gas fraction thereafter).

roughness_fuel	The roughness of the fuel surface.
roughness_clad	The roughness of the cladding surface.
roughness_coef	The coefficient for the roughness summation.
interaction_layer	One of 0 (fuel-cladding chemical interaction layer not considered) and 1 (interaction layer considered).
slave	The boundary id for the slave surface.
stefan_boltzmann	The Stefan-Boltzmann constant.
tangential_tolerance	Tangential distance to extend edges of contact surfaces.
variable	**Required**. The temperature variable name.

12 AuxKernels

AuxKernels are used to compute values for AuxVariables. They often compute quantities based on functions, solution variables, and material properties. AuxKernels can apply to blocks or boundaries. If not block or boundary is specified, the AuxKernel applies to the entire model.

```
[AuxKernels]
  [./name]
    type = <AuxKernel type>
    block = <string list>
    boundary = <string list>
    ...
  [../]
[]
```

type	Type of auxiliary kernel.
block	List of blocks. Either block numbers or names.
boundary	List of boundaries (side sets). Either boundary numbers or names.

12.1 AuxKernels for Output

12.1.1 EutecticThicknessFCCI

The EutecticThicknessFCCI AuxKernel is used to calculated the eutectic penetration thickness when the temperature is above the eutectic melting point for Fuel-Clad Chemical Interaction of metal fuels with cladding. The boundary temperature is required. The rate of penetration is added to the variable tracking the penetration thickness for each time step. The penetration thickness may only grow or remain the same.

```
  [./eutecticthicknessfcci]
    type = EutecticThicknessFCCI
    boundary = <string list>
    variable = <variable>
    temperature = <variable>
    eutectic_melt = <real> (988.15)
    unit_factor = <real> (1.0)
  [../]
```

type	EutecticThicknessFCCI

boundary	**Required**. Boundary where the eutectic penetration thickness is calculated.
variable	**Required**. Name of `AuxVariable` that holds the penetration thickness value.
temperature	**Required**. Name of `Variable` that holds the temperature value.
eutectic_melt	Temperature at which the eutectic melts.
unit_factor	Factor multiplying the rate which may be used to change the units from meters per second.

12.1.2 MaterialRealAux

The `MaterialRealAux` AuxKernel is used to output material properties. Typically, the AuxVariable computed by `MaterialTensorAux` will be an element-level, constant variable. The computed value will be the volume-averaged quantity over the element.

```
[./materialrealaux]
  type = MaterialRealAux
  property = <material property>
  variable = <variable>
[../]
```

type	MaterialRealAux
property	**Required**. Name of material property.
variable	**Required**. Name of `AuxVariable` that will hold result.

12.1.3 MaterialTensorAux

The `MaterialTensorAux` AuxKernel is used to output quantities related to second-order tensors used as material properties. Stress and strain are common examples of these tensors. The AuxKernel allows output of specific tensor entries or quantities computed from the entire tensor. Typically, the AuxVariable computed by `MaterialTensorAux` will be an element-level, constant variable. By default, the computed value will be the volume-averaged quantity over the element. If the parameter qp_select is set to the value of an integration point number (0, 1, ..., n), the computed valued will be the value at that integration point.

```
[./materialtensoraux]
  type = MaterialTensorAux
  tensor = <material property tensor>
  variable = <variable>
  index = <integer>
  quantity = <string>
  point1 = <vector> (0, 0, 0)
  point2 = <vector> (0, 1, 0)
  qp_select = <integer> (0, 1, ..., n)
[../]
```

type	MaterialTensorAux
tensor	**Required**. Name of second-order tensor material property. A typical second-order tensor material property is stress.
variable	**Required**. Name of AuxVariable that will hold result.
index	Index into tensor, from 0 to 5 (xx, yy, zz, xy, yz, zx). Either index or quantity must be specified.
quantity	One of VonMises, PlasticStrainMag, Hydrostatic, Hoop, Radial, Axial, MaxPrincipal, MedPrincipal, MinPrincipal, FirstInvariant, SecondInvariant, ThirdInvariant, or TriAxiality. Either index or quantity must be specified.

12.1.4 ThicknessLayerFCCI

The ThicknessLayerFCCI AuxKernel is used to calculated the interaction layer thickness during Fuel-Clad Chemical Interaction for metal fuels. The mass flux through the boundary is required from a postprocessor. The change in thickness layer is then applied to the variable storing the thickness layer on the boundary. The boundary may be an internal boundary. The growth of the interaction layer assumes the same sign as the normal mass flux. Negative thickness values correspond to negative flux values with respect to the boundary normal. If the boundary normal is in the same direction as the mass flux, the thickness layer would be positive as well. The GapHeatTransfer thermal contact model may be used to provide gap species diffusion to calculate mass flux across gap sections.

```
[./thicknesslayerfcci]
  type = ThicknessLayerFCCI
  boundary = <string list>
  variable = <variable>
  postproc_flux = <string>
  initial_contact_time = <real> (0)
  method = <integer>
  layer_density = <real> (7700)
  layer_mol_weight = <real> (1.215)
  solubility_species = <variable>
  solubility_fuel = <real> (0)
[../]
```

type	ThicknessLayerFCCI
boundary	**Required**. Boundary where the interaction layer thickness is calculated.
variable	**Required**. Name of AuxVariable that holds the thickness layer value.
postproc_flux	**Required**. Name of the Postprocessor computing the mass flux through the boundary.

initial_contact_time	The thickness layer is only calculated after this time value has passed.
method	**Required**. 1 for the correlation with density and molecular weight. 2 for the correlation with solubility limits.
layer_density	*Method 1*. Interaction layer density.
layer_mol_density	*Method 1*. Interaction layer molecular weight.
solubility_species	*Method 2*. Name of Variable for solubility fractions.
solubility_fuel	*Method 2*. Solubility limit in the fuel.

12.2 AuxKernels for Specifying Fission Rate

Note that these AuxKernels are not needed if the Burnup block (see Chapter 13) is present.

12.2.1 FissionRateAux

The FissionRateAux AuxKernel simply sets the value of a variable that stores the fission rate (fissions/m^3/s) to either a constant value or a value prescribed by a function. If both function and value are provided, value is used as a scaling factor on the function.

```
[./fissionrateaux]
  type = FissionRateAux
  variable = <string>
  block = <string list>
  function = <string>
  value = <real>
  variable = <string>
[../]
```

type	FissionRateAux
variable	**Required**. Name of AuxVariable that will hold fission rate. Typically fission_rate.
value	Value of fission rate. If function is present, value is multiplied by the function value.
function	Function describing the fission rate.

12.2.2 FissionRateAuxLWR

FissionRateAuxLWR is designed to calculate fission rate given rod averaged linear power and pellet dimensions.

```
[./fissionrateauxlwr}
  type = FissionRateAuxLWR
  value = <real> (1)
  rod_ave_lin_pow= <string>
```

```
   axial_power_profile = <string>
   pellet_diameter = <real>
   pellet_inner_diameter = <real> (0)
   fuel_volume_ratio = <real> (1)
   energy_per_fission = <real> (3.28451e-11)
   variable = <string>
 [../]
```

value	Fission rate if rod_ave_lin_pow is not present. Scale factor if rod_ave_lin_pow is given.
variable	**Required**. Name of AuxVariable that will hold fission rate. Typically fission_rate.
rod_ave_lin_pow	Function describing rod averaged linear power. This power is the total power, the power from the volumetric fission rate times the volume of fuel times the energy per fission.
axial_power_profile	Function describing axial power profile.
pellet_diameter	**Required**. The diameter of the fuel.
pellet_inner_diameter	The inner diameter of the fuel.
fuel_volume_ratio	Reduction factor for deviation from right circular cylinder fuel. The ratio of actual volume to right circular cylinder volume.
energy_per_fission	The energy released per fission in J/fission.

12.2.3 FissionRateFromPowerDensity

Like FissionRateAux, the FissionRateFromPowerDensity AuxKernel sets the fission rate based on a function and a scaling factor. This AuxKernel is intended to be used specifically in the case where the input function defines the power density (in W/m^3). The power density is divided by user-provided constant that defines the energy per fission (J/fission) to provide the fission rate in (fissions/m^3/s).

```
 [./fissionratefrompowerdensity]
   type = FissionRateFromPowerDensity
   variable = <string>
   block = <string list>
   function = <string>
   energy_per_fission = <real>
 [../]
```

type	FissionRateAux
variable	**Required**. Name of AuxVariable that will hold fission rate. Typically fission_rate.
function	**Required**. Function describing the power density in W/m^3.
energy_per_fission	**Required**. Energy released per fission in J/fission.

50

12.3 Other AuxKernels

12.3.1 Al2O3Aux

```
[./al2o3aux]
  type = Al2O3Aux
  variable = <string>
  function = <string>
  model = <string> (function)
  temp = <string>
[../]
```

type   Al2O3Aux

variable **Required**. Variable name corresponding to the Al2O3 thickness.

function Function describing the Al2O3 thickness as a function of time.

model   One of function or griess. The griess option invokes a correlation appropriate for plate fuel.

temp   Variable name for temperature variable. Typically `temp`.

12.3.2 BurnupAux

BurnupAux computes burnup given the fission rate. Note that this AuxKernel is not needed if the Burnup block (see Chapter 13) is present.

```
[./burnupaux]
  type = BurnupAux
  fission_rate = <string>
  density = <real>
  molecular_weight = <real> (0.270)
[../]
```

type   BurnupAux

variable   **Required**. Variable name corresponding to the burnup. Typically `burnup`.

fission_rate   **Required**. Variable name corresponding to the fission rate. Typically `fission_rate`.

density   **Required**. The initial fuel density.

molecular_weight The molecular weight.

12.3.3 FastNeutronFluenceAux

```
[./fastneutronfluenceaux]
  type = FastNeutronFluenceAux
```

```
    variable = <string>
    fast_neutron_flux = <string>
  [../]
```

type	FastNeutronFluenceAux
variable	**Required**. Variable name corresponding to the fast neutron fluence. Typically fast_neutron_fluence.
fast_neutron_flux	**Required**. Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.

12.3.4 FastNeutronFluxAux

```
  [./fastneutronfluxaux]
    type = FastNeutronFluxAux
    variable = <string>
    rod_ave_lin_pow = <string>
    axial_power_profile = <string>
    factor = <real>
    function = <string>
    q_variable = <string>
  [../]
```

type	FastNeutronFluxAux
variable	**Required**. Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.
rod_ave_lin_pow	Function describing rod averaged linear power. This power is the total power, the power from the volumetric fission rate times the volume of fuel times the energy per fission.
axial_power_profile	Function describing axial power profile.
factor	The fast neutron flux if function, rod_ave_lin_pow, or q_variable is not given. Otherwise, a scale factor. Recommended scale factor value is 3e13 $(n/(m^2\text{-}s)/(W/m))$.
function	Function that describes the fast neutron flux.
q_variable	Variable holding linear heat rate in pellet in W/m.

Only one of function, rod_ave_lin_pow, and q_variable may be given.

12.3.5 GrainRadiusAux

The GrainRadiusAux model is a simple empirical model for calculating grain growth. This can be used with the Sifgrs model (17.1).

```
  [./grainradiusaux]
```

52

```
    type = GrainRadiusAux
    variable = <string>
    temp = <string>
  [../]
```

type	GrainRadiusAux
variable	**Required**. Variable name corresponding to the fuel grain radius.
temp	**Required**. Variable name for temperature variable. Typically `temp`.

12.3.6 OxideAux

```
  [./oxideaux]
    type = OxideAux
    variable = <string>
    fast_neutron_flux = <string>
    lithium_concentration = <real> (0)
    model_option = <int> (1)
    oxide_scale_factor = <real> (1)
    tin_content = <real> (1.38)
    temperature = <string>
    use_coolant_channel = <bool> (false)
    start_time = <real> (0)
    end_time = <real> (inf)
```

type	OxideAux
variable	**Required**. Variable name corresponding to the zirconia thickness.
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically `fast_neutron_flux`.
lithium_concentration	Lithium concentration in ppm.
model_option	If 1, uses the EPRI KWU CE model. Otherwise, uses the EPRI SLI model.
oxide_scale_factor	Scale factor applied to the rate of oxide growth.
tin_content	Tin content in wt%.
temperature	**Required**. Variable name for temperature variable. Typically `temp`.
use_coolant_model	If true, model will adjust surface temperature based on the coolant channel model.
start_time	Start the oxide growth at a specific time.
end_time	End the oxide growth at a specific time, for example in dry storage.

53

12.3.7 FeCrAlOxideAux

```
[./fecraloxideaux]
  type = FeCrAlOxideAux
  variable = <string>
  parabolic_rate_constant = <real> (784.0)
  activation_energy = <real> (41373.7)
  oxide_scale_factor = <real> (1)
  temperature = <string>
  start_time = <real> (0)
  end_time = <real> (inf)
[../]
```

type	FeCrAlOxideAux
variable	**Required**. Variable name corresponding to the oxide thickness.
parabolic_rate_constant	The constant preceding the exponential term in the parabolic rate equation. Needs to be entered in units of kg^2/m^4-s.
acitivation_energy	The activation energy for oxidation divided by the ideal gas constant. Entered in units of K.
oxide_scale_factor	Scale factor applied to the rate of oxide growth.
temperature	**Required**. Variable name for temperature variable. Typically temp.
start_time	Start the oxide growth at a specific time.
end_time	End the oxide growth at a specific time, for example in dry storage.

12.3.8 PelletIdAux

PelletIdAux is used to compute a pellet number. It may be used with a discrete pellet or smeared fuel column mesh.

```
[./pelletidaux]
  type = PelletIdAux
  variable = <string>
  a_lower = <real>
  a_upper = <real>
  number_pellets = <integer>
[../]
```

type	PelletIdAux
variable	**Required**. AuxVariable name corresponding to the Pellet ID.
a_lower	**Required**. The lower axial coordinate of the fuel stack.

a_upper   **Required**. The upper axial coordinate of the fuel stack.

number_pellets **Required**. Number of fuel pellets.

13 Burnup

The `Burnup` block computes fission rate and burnup for LWR fuel including the radial power factor. It is not appropriate for other fuel configurations. Use of the `Burnup` block will cause BISON to create and populate `burnup`, `fission_rate`, and optionally other `AuxVariables`.

The radial power factor calculation is performed on a secondary numerical grid, created internally by BISON. This is the reason for the `num_radial` and `num_axial` line commands. Once the fission rate, burnup, and other quantities are computed on this secondary grid, they are mapped back to the finite element mesh.

```
[Burnup]
  [./burnup]
    block = <string list>
    rpf_active = <bool> (true)
    fuel_type = <string> (UO2)
    reactor_type = <string> (LWR)
    include_gadolinia = <bool> (false)
    rod_ave_linear_power = <string>
    axial_power_profile = <string>
    num_radial = <integer>
    num_axial = <integer>
    fuel_pin_geometry = <string>
    a_lower = <real>
    a_upper = <real>
    fuel_inner_radius = <real> (0)
    fuel_outer_radius = <real> (0.0041)
    fuel_volume_ratio = <real> (1)
    density = <real>
    energy_per_fission = <real> (3.28451e-11)
    p1 = <real> (3.45)
    i_enrich = <real list> (0.05, 0.95, 0, 0, 0, 0)
    wtfract_gadolinia = <real>
    sigma_c = <real list> (9.7, 0.78, 58.6, 100, 50, 80)
    sigma_f = <real list> (41.5, 0, 105, 0.584, 120, 0.458)
    sigma_a_thermal = <real list> (359.68, 1.56, 1207.5,
                                   193.5, 1095.24, 11.11)
    N235 = <string>
    N238 = <string>
    N239 = <string>
    N240 = <string>
    N241 = <string>
    N242 = <string>
    N155 = <string>
```

```
    N157 = <string>
    RPF = <string>
  [../]
[]
```

block	**Required**. List of fuel blocks. Either block numbers or names.
rpf_active	Flag for turning calculation of radial power factor on.
fuel_type	Fuel type. One of UO2 or U3Si2. Will set defaults for sigma_f and sigma_c in conjunction with the reactor type.
reactor_type	Reactor type. One of LWR or HWR. Will set default values for p1, sigma_f, and sigma_c if those are not otherwise specified.
include_gadolinia	Flag for turning on calculation of gadolinia (Gd_2O_3) effects.
rod_ave_lin_pow	Function describing rod averaged linear power. This power is the total power, the power from the volumetric fission rate times the volume of fuel times the energy per fission.
axial_power_profile	Function describing axial power profile.
num_radial	Number of radial divisions in secondary grid used to compute radial power profile.
num_axial	Number of axial divisions in secondary grid used to compute radial power profile.
fuel_pin_geometry	Name of the FuelPinGeometry object (see 23.1).
a_lower	**Required if fuel_pin_geometry is not specified**. The lower axial coordinate of the fuel stack.
a_upper	**Required if fuel_pin_geometry is not specified**. The upper axial coordinate of the fuel stack.
fuel_inner_radius	The inner radius of the fuel.
fuel_outer_radius	The outer radius of the fuel.
fuel_volume_ratio	Reduction factor for deviation from right circular cylinder fuel. The ratio of actual volume to right circular cylinder volume.
density	**Required**. The initial fuel density.
energy_per_fission	The energy released per fission in J/fission.
p1	Distribution function coefficient p1. If not given, will take default value based on reactor_type.
i_enrich	The initial enrichment for the six isotopes.
wtfract_gadolinia	**Required if include_gadolinia=true**. Initial weight fraction of Gd_2O_3.
sigma_c	The capture cross sections for the considered isotopes (6 by default, 8 if include_gadolinia=true). If not given, will take default value based on reactor_type.

`sigma_f`	The fission cross sections for the considered isotopes (6 by default, 8 if `include_gadolinia=true`). If not given, will take default value based on `reactor_type`.
`sigma_a_thermal`	The thermal absorption cross sections for the considered isotopes (6 by default, 8 if `include_gadolinia=true`).
`N235`	Indicates that the output of the concentration of U-235 is required. Typically N235.
`N238`	Indicates that the output of the concentration of U-238 is required. Typically N238.
`N239`	Indicates that the output of the concentration of Pu-239 is required. Typically N239.
`N240`	Indicates that the output of the concentration of Pu-240 is required. Typically N240.
`N241`	Indicates that the output of the concentration of Pu-241 is required. Typically N241.
`N242`	Indicates that the output of the concentration of Pu-242 is required. Typically N242.
`N155`	Indicates that the output of the concentration of Gd-155 is required. Typically N155. Use only with `include_gadolinia=true`.
`N157`	Indicates that the output of the concentration of Gd-157 is required. Typically N157. Use only with `include_gadolinia=true`.
`RPF`	Indicates that the output of the radial power factor is required. Typically RPF.

14 Kernels

`Kernels` are used to evaluate integrals associated with a given term in a PDE. They often compute quantities based on functions, solution variables, auxiliary variables, and material properties. All `Kernels` act on blocks. If no block is specified, the `Kernel` will act on the entire model.

```
[Kernels]
  [./name]
    type = <kernel type>
    block = <string list>
    ...
  [../]
[]
```

type   Type of kernel.

block   List of blocks. Either block numbers or names.

14.1 Arrhenius Diffusion

Kernel for applying an Arrhenius diffusion term. If present, an `ArrheniusDiffusionCoef` material model must also be present.

```
  [./arrheniusdiffusion]
    type = ArrheniusDiffusion
    variable = <variable>
  [../]
```

type   `ArrheniusDiffusion`

variable   **Required**. Variable associated with this volume integral.

14.2 BodyForce

Kernel for applying an arbitrary body force to the model.

```
  [./bodyforce]
    type = BodyForce
    variable = <variable>
    value = <real> (0)
```

```
    function = <string> (1)
  [../]
```

`type`	BodyForce
`variable`	**Required**. Variable associated with this volume integral.
`value`	Constant included in volume integral. Multiplied by the value of `function` if present.
`function`	Function to be multiplied by `value` and used in the volume integral.

14.3 Gravity

Gravity may be applied to the model with this kernel. The required density is computed and provided internally given inputs in the `Materials` block.

```
  [./gravity]
    type = Gravity
    variable = <variable>
    value = <real> (0)
  [../]
```

`type`	Gravity
`variable`	**Required**. Variable name corresponding to the displacement direction in which the gravity load should be applied.
`value`	Acceleration of gravity. Typically -9.81 (m/s^2).

14.4 Heat Conduction

Kernel for diffusion of heat or divergence of heat flux.

```
  [./heatconduction]
    type = HeatConduction
    variable = <variable>
  [../]
```

`type`	HeatConduction
`variable`	**Required**. Variable name corresponding to the heat conduction equation. Typically `temp`.

14.5 Heat Conduction Time Derivative

Kernel for $\rho C_p \partial T / \partial t$ term of the heat equation.

```
[./heatconductiontimederivative]
  type = HeatConductionTimeDerivative
  variable = <variable>
[../]
```

type	HeatConductionTimeDerivative
variable	**Required**. Variable name corresponding to the heat conduction equation. Typically `temp`.

14.6 Heat Source

The `HeatSource` kernel applies a volumetric heat source to specified blocks within the model. Built on the `BodyForce` kernel's code, the `HeatSource` kernel provides a more relevant name for easier input-file specification.

```
[./heatsource]
  type = HeatSource
  variable = <variable>
  value = <real> (1)
  function = <string> (1)
  block = <string list>
[../]
```

type	HeatSource
variable	**Required**. The variable associated with the heat source.
value	Value of the heat source; will be multiplied by the optional `function`.
function	The function describing the volumetric heat source.
block	The list of block id's (SubdomainID) to which the heat source will be applied.

14.7 Isotropic Diffusion

`IsotropicDiffusion` is just like `ArrheniusDiffusion` except that it takes an arbitrary material property and uses it as the diffusivity. For example, it could be coupled to the material property `ArrheniusDiffusionCoef` using the material property "arrhenius_diffusion_coef" or to `ArrheniusMaterialProperty` using any name for the diffusivity.

```
[./diffusion]
  type = IsotropicDiffusion
  variable = <variable>
  diffusivity_property = <string> (diffusivity)
[../]
```

`type`	`IsotropicDiffusion`
`variable`	**Required**. Variable associated with this volume integral.
`diffusivity_property`	The name of the material property to be used as the diffusivity.

14.8 Neutron Heat Source

Kernel for the volumetric heat source associated with fission.

```
[./neutronheatsource]
  type = NeutronHeatSource
  variable = <variable>
  burnup_function = <string>
  fission_rate = <variable>
  decay_heat_function = <string>
  fuel_pin_geometry = <string>
  outer_diameter = <real>
  inner_diameter = <real> (0)
  area = <real>
[../]
```

`type`	`NeutronHeatSource`
`variable`	**Required**. Variable name corresponding to the heat conduction equation. Typically `temp`.
`burnup_function`	Name of the `Burnup` sub-block, if any (typically `burnup`). May not be used with `fission_rate`. Use of `burnup_function` is preferred.
`fission_rate`	Variable name corresponding to the fission rate. Typically `fission_rate`.
`decay_heat_function`	Name of the postprocessor giving the decay heat curve. Typically supplied for LOCA simulations.
`fuel_pin_geometry`	Name of the FuelPinGeometry object (see 23.1).
`outer_diameter`	Outer diameter of the cold fuel pellets if `fuel_pin_geometry` is not specified.
`inner_diameter`	Inner diameter of the cold fuel pellets if `fuel_pin_geometry` is not specified.
`area`	Cross-sectional area of the cold fuel pellets if `fuel_pin_geometry` or the diameters are not specified.

14.9 SolidMechanics

The `SolidMechanics` block specifies inputs for the divergence of stress as part of the equations of solid mechanics. The divergence of stress is a `Kernel` in MOOSE nomenclature. The

`SolidMechanics` block informs MOOSE of the divergence kernels but is not placed inside the `Kernels` block in the input file.

```
[SolidMechanics]
  [./solidmechanics]
    disp_x = <variable>
    disp_y = <variable>
    disp_z = <variable>
    disp_r = <variable>
    temp = <variable>
  [../]
[]
```

disp_x   Variable name for displacement variable in x direction. Typically `disp_x`.

disp_y   Variable name for displacement variable in y direction. Typically `disp_y`.

disp_z   Variable name for displacement variable in z direction. Typically `disp_z` for 3D and `disp_y` for axisymmetric models.

disp_r   Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically `disp_x`.

temp   Variable name for temperature variable. Necessary for thermal expansion. Typically `temp`.

14.10 Thermo-diffusion (Soret effect, thermophoresis)

`ThermoDiffusion` is used to model mass flux of the form

$$J = -\frac{DQC}{RT^2}\nabla T \tag{14.1}$$

where D is the mass diffusivity (property name is "mass_diffusivity"), Q is the heat of transport, C is the concentration, R is the gas constant, and T is the temperature.

```
  [./soret_diffusion]
    type = ThermoDiffusion
    variable = <variable>
    temp = <variable>
    gas_constant = <real> (8.31446)
  [../]
```

type   ThermoDiffusion

variable   **Required**. Variable associated with this volume integral.

temp   **Required**. Coupled temperature variable.

gas_constant   Universal gas constant.

14.11 TimeDerivative

Kernel for applying a time rate of change term ($\partial u / \partial t$) to the model.

```
[./timederivative]
  type = TimeDerivative
  variable = <variable>
[../]
```

type	TimeDerivative
variable	**Required**. Variable associated with this volume integral.

15 Hydride formation in the cladding

Modeling the evolution of hydrides in the cladding requires two variables to track concentration of the hydrogen in solution and the hydrogen as hydride, kernels that act as source/sink terms for the concentration variables, kernels for Fickian and thermal diffusion of the hydrogen, a kernel for the time derivative of hydrogen, and a set of material models that calculate precipitation and dissolution rates, and the heat of transport and diffusivity of hydrogen. In addition, if hydrogen pickup is of interest, a corrosion model and pickup boundary condition must be added. However, there is also an Action that sets up everything except for the two variables and the hydrogen pickup; we cover the variables and the Action first.

15.1 Using the CladdingHydrides Action (the easy way)

The two concentration variables track the hydrogen in solid solution (commonly referred to as C_{ss}) and the equivalent concentration of hydrogen bound in the precipitated hydrides (commonly referred to as C_p). These concentrations are usually specified in ppm by weight. Since the hydride may have a steep gradient, monomials are helpful to keep the concentration positive. Also note below the large scalings that are useful for speeding convergence.

```
[Variables]
  [./hydrogen_in_solution_ppm]
    scaling = 1e12
    block = 'clad'
  [../]
  [./hydrogen_as_hydride_ppm]
    order = CONSTANT
    family = MONOMIAL
    scaling = 1e12
    block = 'clad'
  [../]
[]
```

The `CladdingHydrides` Action adds Fick's Law, the Soret effect, a precipitation/dissolution sink/source term, and a specialized time derivative for soluble hydrogen. It also adds a precipitation/dissolution source/sink term and a time derivative for hydrogen as hydride. These kernels require a few specialized material models for the actual precipitation/dissolution rate, the mass diffusivity of hydrogen in the matrix, and the heat of transport for hydrogen; these are also added by the Action. Thus, if you use this Action and have the variables listed above, then you only need to add a hydrogen pickup boundary condition (see 10.7) to get the complete hydride model for the cladding.

```
[CladdingHydrides]
  [./some_name]
    block = <list of blocks>
    temperature = <string>
    hydrogen_in_solution_ppm = <string>
    hydrogen_as_hydride_ppm = <string>
    hydride_clamp = <real> (1)
    diffusivity_frequency_factor = <real> (0.8e-7)
    diffusivity_activation_energy = <real> (33306)
    heat_of_transport = <real> (25121)
  [../]
[]
```

block	List of blocks where the hydride model is to be applied (the cladding blocks).
temperature	**Required**. Name of the temperature variable.
hydrogen_in_solution_ppm	**Required**. Name of the variable for hydrogen in solution (wt.ppm).
hydrogen_as_hydride_ppm	**Required**. Name of the variable for hydrogen as hydrides (wt.ppm).
hydride_clamp	The limiting value of the volume fraction of hydrides. This is used to control creation of a thick hydride rim. Must be in [0,1].
diffusivity_frequency_factor	The Arrhenius coefficient for hydrogen mass diffusivity in the cladding.
diffusivity_activation_energy	The Arrhenius activation energy for hydrogen mass diffusivity in the cladding.
heat_of_transport	The heat of transport for dissolved hydrogen in the cladding.

15.2 Using individual kernels and materials (the hard way)

15.2.1 HydridePrecipitationRate

A single material is used to calculate the precipitation or dissolution rate. This is not a material property *per se*; it is just used this way for convenience.

```
[Materials]
  [./precip_rate]
    type = HydridePrecipitationRate
    block = <string>
    temp = <variable>
    hydrogen_in_solution_ppm = <variable>
```

```
    hydrogen_as_hydride_ppm = <variable>
    hydride_clamp = <real> (1)
  [../]
[]
```

`type`	`HydridePrecipitationRate`
`block`	The volume associated with this material.
`temp`	**Required**. Coupled temperature variable.
`hydrogen_in_solution_ppm`	**Required**. Coupled C_{ss} in wt.ppm.
`hydrogen_as_hydride_ppm`	**Required**. Coupled C_p in wt.ppm.
`hydride_clamp`	The limiting value of the volume fraction of hydrides. This is used to control creation of a thick hydride rim. Must be in [0,1].

The clamping feature is used to limit the amount of hydride that forms by precipitation. If the hydride concentration exceeds the clamp value, the local rate of hydride precipitation will drop to zero even if there is local over saturation of hydrogen in solid solution.

15.2.2 HydrogenSource and HydrideSource

Finally, we just need to add two source kernels: one for C_{ss} and one for C_p. The kernel is a source if precipitation increases the concentration or a sink if precipitation decreases the concentration (i.e. precipitation is a sink for hydrogen in solid solution). For convenience, these are split into `HydrogenSource` and `HydrideSource` so that the user does not have to keep track of the sign on the source term.

```
[Kernels]
  [./hydride_source]
    type = HydrideSource
    variable = <variable>
  [../]
[]
```

`type`	`HydrideSource`
`variable`	**Required**. The concentration of hydrogen in one of the phases.

`HydrogenSource` has the same form as `HydrideSource` (only the type changes).

15.2.3 HydrogenTimeDerivative

Due to the special treatment of the hydrogen in solution (a volume fraction term appears in the time derivative), we need a special kernel for this term. It needs to couple into the hydride variable as well.

```
[Kernels]
  [./hydrogen_dHdt]
    type = HydrogenTimeDerivative
    variable = <variable>
    hydrogen_as_hydride_ppm = <string>
  [../]
[]
```

`type`	HydrogenTimeDerivative
`variable`	**Required**. The concentration of hydrogen in solution (wt.ppm).
`hydrogen_as_hydride_ppm`	**Required**. The concentration of hydrogen as hydrides (wt.ppm).
`hydride_clamp`	The limiting value of the volume fraction of hydrides. This is used to control creation of a thick hydride rim. Must be in [0,1].

15.2.4 HydrogenDiffusivity

Similarly to `HydrogenTimeDerivative`, the diffusion term for the hydrogen in solution must also account for the phase fraction of alloy that is available for diffusion.

```
[Materials]
  [./hydrogen_diffusivity]
    type = HydrogenDiffusivity
    frequency_factor = <real>
    activation_energy = <real>
    temp = <string>
    property_name = <string>( 'mass_diffusivity' )
    hydrogen_as_hydride_ppm = <string>
    hydride_clamp = <real> (1)
  [../]
[]
```

`type`	HydrogenDiffusivity
`frequency_factor`	Arrhenius frequency factor.
`activation_energy`	Arrhenius activation energy.
`temp`	**Required**. The coupled temperature.
`property_name`	The named property that will be declared by this material.
`hydrogen_as_hydride_ppm`	**Required**. The concentration of hydrogen as hydrides (wt.ppm).

`hydride_clamp`	The limiting value of the volume fraction of hydrides. This is used to control creation of a thick hydride rim. Must be in [0,1].

15.2.5 Other things needed for modeling hydrides without the CladdingHydrides Action

The kernels and materials for simulating the transport of the hydrogen in solution are generic kernels and are not covered here; see 14.7 for mass diffusion and 14.10 for the Soret effect. These two kernels require material properties for mass diffusivity (use `HydrogenDiffusivity`) and heat of transport of hydrogen (use `GenericConstantMaterial` and name the property "heat_of_transport") in the alloy. Also not shown here is the flux boundary condition for hydrogen pickup at the oxide interface (see 10.7).

16 Materials

The `Materials` block is for specifying material properties and models.

```
[Materials]
  [./name]
    type = <material type>
    block = <string list>
    ...
  [../]
[]
```

type   Type of material model

block   List of blocks. Either block numbers or names.

16.1 Thermal Models

16.1.1 HeatConductionMaterial

`HeatConductionMaterial` is a general-purpose material model for heat conduction. It sets the thermal conductivity and specific heat at integration points.

```
  [./heatconductionmaterial]
    type = HeatConductionMaterial
    thermal_conductivity = <real>
    thermal_conductivity_x = <string>
    thermal_conductivity_y = <string>
    thermal_conductivity_z = <string>
    thermal_conductivity_temperature_function = <string>
    specific_heat = <real>
    specific_heat_temperature_function = <string>
  [../]
```

type	HeatConductionMaterial
thermal_conductivity	Thermal conductivity.
thermal_conductivity_x	Thermal conductivity Postprocessor for the x direction.
thermal_conductivity_y	Thermal conductivity Postprocessor for the y direction.

`thermal_conductivity_z`	Thermal conductivity `Postprocessor` for the z direction.
`thermal_conductivity_temperature_function`	Function describing thermal conductivity as a function of temperature.
`specific_heat`	Specific heat.
`specific_heat_temperature_function`	Function describing specific heat as a function of temperature.

16.1.2 ThermalCladMaterial

The `ThermalCladMaterial` model computes the specific heat and thermal conductivity for a variety of exotic cladding materials. The choices are `Thermal316`, `ThermalAlloy33`, `ThermalD9`, `ThermalHT9`, `ThermalIrradiationCreepHT9`, `ThermalMo`, and `ThermalNa`. The details of these models are described in the Theory Manual. Examples of their use can be found in /bison/tests/thermalTests/, /bison/tests/thermalD9/, /bison/tests/thermalNa/, and /bison/tests/thermalHT9/.

```
[./thermalCladMaterial]
  type = Thermal<string>
  block = <string list>
  temp = <string>
[../]
```

type   `Thermal<string>`, where `<string>` is the fuel material type (eg. HT9)

block   List of blocks this material applies to.

temp   Name of temperature variable. Typically `temp`.

16.1.3 ThermalFeCrAl

The `ThermalFeCrAl` model computes the specific heat and thermal conductivity for a variety of FeCrAl cladding alloys. The alloys to choose from are Special Metals Incoloy MA956, Plansee PM2000, Kanthal APMT, Resistalloy International Fecralloy and ORNL's C35M.

```
[./thermalFeCrAl]
  type = ThermalFeCrAl
  block = <string list>
  material = <string> (APMT)
  temp = <string>
[../]
```

type   `ThermalFeCrAl`

block   List of blocks this material applies to.

material   Name of the chosen FeCrAl alloy. Choices are MA956, PM2000, APMT, FE-CRALLOY or C35M

16.1.4 ThermalFuel

The `ThermalFuel` model computes specific heat and thermal conductivity for oxide fuel. A number of correlations are available.

```
[./thermalfuel]
  type = ThermalFuel
  temp = <string>
  burnup_function = <string>
  burnup = <string>
  porosity = <string>
  initial_porosity = <real> (0.05)
  oxy_to_metal_ratio = <real> (2.0)
  Pu_content = <real> (0.0)
  Gd_content = <real> (0.0)
  model = < 0, 1, 2, 3, 4, or 5 for
        Duriez, Amaya, Fink-Lucuta, Halden, NFIR, or Modified NFIR >
[../]
```

type	ThermalFuel
temp	Name of temperature variable. Typically `temp`.
burnup_function	Name of the `Burnup` sub-block, if any (typically `burnup`). May not be used with the `burnup` line command. Use of `burnup_function` is preferred.
burnup	Name of burnup variable. Typically `burnup`.
porosity	Name of porosity variable. Typically `porosity`. Optional.
initial_porosity	Initial porosity.
oxy_to_metal_ratio	Ratio of oxygen atoms to metal atoms.
Pu_content	Weight fraction of Pu in MOX fuel (typically 0.07).
Gd_content	Weight fraction of Gd in fuel.
model	**Required**. The chosen thermal conductivity model.

16.1.5 ThermalFastMOX

The `ThermalFastMOX` model computes the thermal conductivity for fast MOX fuel. Currently a single correlation is available from Inoue and used by Karahan. Details are provided in the theory manual. The specific heat capacity is treated as a constant value for now.

```
[./thermalfastmox]
  type = ThermalFastMOX
  temp = <string>
  burnup_function = <string>
```

```
    burnup = <string>
    porosity = <string>
    initial_porosity = <real> (0.05)
    oxy_to_metal_ratio = <real> (2.0)
  [../]
```

type	ThermalFastMOX
temp	Name of temperature variable. Typically `temp`.
burnup_function	Name of the `Burnup` sub-block, if any (typically `burnup`). May not be used with the `burnup` line command. Use of `burnup_function` is preferred.
burnup	Name of burnup variable. Typically `burnup`.
porosity	Name of porosity variable. Typically `porosity`. Optional.
initial_porosity	Initial porosity.
oxy_to_metal_ratio	Ratio of oxygen atoms to metal atoms.

16.1.6 ThermalMAMOX

The `ThermaMAMOX` model computes the thermal conductivity for minor actinide, Np- and Am-bearing, fast MOX fuel (MA-MOX). This model is based on the correlation found here [8]. The correlation takes in to account the oxygen to metal ratio of the fuel, the Am quantity and the Np quantity present. The specific heat capacity is treated as a constant value.

```
  [./thermalMAMOX]
    type = ThermalMAMOX
    temp = <string>
    burnup_function = <string>
    burnup = <string>
    porosity = <string>
    initial_porosity = <real> (0.05)
    oxy_to_metal_ratio = <real> (2.0)
  [../]
```

type	ThermalFastMOX
temp	Name of temperature variable. Typically `temp`.
burnup_function	Name of the `Burnup` sub-block, if any (typically `burnup`). May not be used with the `burnup` line command. Use of `burnup_function` is preferred.
burnup	Name of burnup variable. Typically `burnup`.
porosity	Name of porosity variable. Typically `porosity`. Optional.
initial_porosity	Initial porosity.
oxy_to_metal_ratio	Ratio of oxygen atoms to metal atoms.

16.1.7 ThermalFuelMaterial

The `ThermalFuelMaterial` model computes specific heat and thermal conductivity for a variety of exotic fuel materials. The choices are `ThermalU`, `ThermalU10Mo`, `ThermalU3Si5UN`. The details of these models are described in the Theory Manual. Examples of their use can be found in /bison/tests/thermalTests/.

```
[./thermalFuelMaterial]
  type = Thermal<string>
  block = <string list>
  temp = <string>
  porosity = <string>
  density = <real>
[../]
```

type	`Thermal<string>`, where `<string>` is the fuel material type (eg. U10Mo)
block	List of blocks this material applies to.
temp	Name of temperature variable. Typically `temp`.
porosity	Name of porosity variable. Typically `porosity`. Optional.
density	**Required**. Density, assumed constant.

16.1.8 ThermalUPuZr

The `ThermalUPuZr` model computes the specific heat and thermal conductivity for any alloy of U-Pu-Zr metal fuel. If `porosity` and `porosity_material` are both defined an error will result. The user has the choice of not supplying any porosity, supplying a variable for porosity or couple to a material that calculates the porosity.

In addition, a user can specify molar (X_i) or weight fraction (w_i). It is important to note that molar fraction can be coupled to a variable, while weight fraction remains constant.

```
[./thermalUPuZr]
  type = ThermalUPuZr
  block = <string list>
  temp = <string>
  porosity = <string>
  porosity_material = <bool> (false)
  X_Pu = <string>
  X_Zr = <string>
  pu_weight = <real>
  zr_weight = <real>
  thcond_model = <string> (billone)
  spheat_model = <string> (karahan)
  A_U = <real> (0.2380289)
  A_Pu = <real> (0.244)
  A_Zr = <real> (0.091224)
  k_scalar = <real> (1.0)
```

```
   porosity_factor = <real> (2.5)
 [../]
```

`type`	ThermalUPuZr
`block`	List of blocks this material applies to.
`temp`	Name of temperature variable. Typically `temp`.
`porosity`	Name of porosity variable. Typically `porosity`. Optional.
`porosity_material`	A flag indicating whether porosity is being calculated by a material property (e.g., VSwellingUPuZr). Optional.
`X_Pu`	Name of the molar fraction of plutonium variable. Typically `X_Pu`.
`X_Zr`	Name of the molar fraction of zirconium variable. Typically `X_Zr`.
`pu_weight`	The weight fraction of plutonium in the fuel. This value remains constant. Given as a decimal (e.g., 0.2).
`zr_weight`	The weight fraction of zirconium in the fuel. This value remains constant. Given as a decimal (e.g., 0.15).
`A_U`	Molar mass of uranium in [kg/mol].
`A_Pu`	Molar mass of plutonium in [kg/mol].
`A_Zr`	Molar mass of zirconium in [kg/mol].
`k_scalar`	Multiplication factor multiplied against the thermal conductivity.
`porosity_factor`	Factor used when calculating porosity correction).

16.1.9 ThermalSilicideFuel

The `ThermalSilicideFuel` model computes the specific heat and thermal conductivity for different phases of uranium silicide fuel including pure silicon, pure uranium metal, U_3Si_2, U_3Si, and U_3Si_5. Three models are available for thermal conductivity and two models for specific heat. The details of these models can be found in the theory manual.

```
 [./thermalSilicideFuel]
   type = ThermalSilicideFuel
   block = <string list>
   temp = <string>
   silicon_mole_fraction = <real> (0.4)
   specific_heat_model = <string> (WHITE)
   thermal_conductivity_model = <string> (WHITE)
 [../]
```

`type`	ThermalSilicideFuel
`block`	List of blocks this material applies to.
`temp`	Name of temperature variable. Typically `temp`.

silicon_mole_fraction	The mole fraction of silicon in the fuel. For example for U_3Si_2 this parameter would be 0.4
specific_heat_model	Name of the model to be used to compute the specific heat. Default is WHITE. Options are WHITE and IAEA.
thermal_conductivity_model	Name of the model to be used to compute the thermal conductivity. Default is WHITE. Options are WHITE, SHIMIZU, and ZHANG.

16.2 Solid Mechanics Models

16.2.1 CreepFastMOXModel

`CreepFastMOXModel` is used to model the creep behavior of fast MOX.

```
[./creepfastmox]
  type = CreepFastMOXModel
  absolute_tolerance = <real>
  creeprate_scalef = <real>
  fission_rate = <string>
  fraction_of_temp_outside_model_bounds = <real>
  grain_size = <real>
  max_creep_increment = <real>
  max_its = <int>
  output = <string>
  output_iteration_info = <bool>
  output_iteration_info_on_error = <bool>
  output_properties = <string>
  outputs = <string>
  relative_tolerance = <real>
  stress_free_temperature = <real>
  temp = <string>
  thermal_expansion = <real>
  thermal expansion coefficient = <real>
  thermal_expansion_function = <string>
  thermal_expansion_function_type = <string>
  thermal_expansion_reference_temperature = <real>
  use_displaced_mesh = <bool>
[../]
```

type	CreepFastMOXModel
absolute_tolerance	Absolute convergence tolerance for sub-newtion iteration
creeprate_scalef	scaling factor for creep rate
fission_rate	Coupled fission rate
fraction_of_temp_outside_model_bounds	Fraction of temperature limit outside of model bounds

76

`grain_size`	grain size in micro meters
`max_creep_increment`	Maximum creep strain increment allowed by accuracy time step criterion
`max_its`	Maximum number of sub-newton iterations
`output`	The reporting postprocessor to use for the max_iterations value
`output_iteration_info`	Set true to output sub-newton iteration information
`output_iteration_info_on_error`	Set true to output sub-newton iteration information when a step fails
`output_properties`	List of material properties, from this material, to output (outputs must also be defined to an output type)
`outputs`	Vector of output names were you would like to restrict the output of variables(s) associated with this object
`relative_tolerance`	Relative convergence tolerance for sub-newton iteration
`stress_free_temperature`	The stress-free temperature. If not specified, the initial temperature is used.
`temp`	Coupled Temperature
`thermal_expansion`	The thermal expansion coefficient
`thermal_expansion_function`	Thermal expansion coefficient as a function of temperature
`thermal_expansion_function_type`	Type of thermal expansion function. Choices are: instantaneous mean
`thermal_expansion_reference_temperature`	Reference temperature for mean thermal expansion function

16.2.2 CreepPyC

`CreepPyC` is used to model the creep behavior of pyrolytic carbon.

```
[./creeppyc]
  type = CreepPyC
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  flux = <string>
  density = <real>
```

```
    youngs_modulus = <real>
    poissons_ratio = <real>
    thermal_expansion = <real> (0)
    stress_free_temperature = <real>
  [../]
```

type	CreepPyC
disp_x	Variable name for displacement variable in x direction. Typically `disp_x`.
disp_y	Variable name for displacement variable in y direction. Typically `disp_y`.
disp_z	Variable name for displacement variable in z direction. Typically `disp_z` for 3D and `disp_y` for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically `disp_x`.
temp	Name of temperature variable. Typically `temp`.
flux	**Required**. Variable name corresponding to the fast neutron flux. Typically `fast_neutron_flux`.
density	**Required**. The initial material density.
thermal_expansion	Coefficient of thermal expansion.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.

16.2.3 CreepSiC

```
 [./creepsic]
   type = CreepSiC
   disp_x = <string>
   disp_y = <string>
   disp_z = <string>
   disp_r = <string>
   temp = <string>
   fast_neutron_flux = <string>
   k_function = <string>
   youngs_modulus = <real>
   poissons_ratio = <real>
   thermal_expansion = <real> (0)
   stress_free_temperature = <real>
 [../]
```

type	CreepSiC

disp_x	Variable name for displacement variable in x direction. Typically `disp_x`.
disp_y	Variable name for displacement variable in y direction. Typically `disp_y`.
disp_z	Variable name for displacement variable in z direction. Typically `disp_z` for 3D and `disp_y` for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically `disp_x`.
temp	Name of temperature variable. Typically `temp`.
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically `fast_neutron_flux`.
k_function	**Required**. Function that takes temperature as input and gives the K coefficient as output.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
thermal_expansion	Coefficient of thermal expansion.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.

`CreepSiC` is used to model the creep behavior of silicon carbide. The relation is

$$\dot{\varepsilon}_{cr} = K\sigma\phi. \tag{16.1}$$

16.2.4 CreepU10Mo

`CreepU10Mo` is used to model the thermal and irradiation creep behavior of U-10Mo fast reactor fuel.

```
[./creepu10mo]
  type = CreepU10Mo
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  fission_rate = <string>
  youngs_modulus = <real>
  poissons_ratio = <real>
  thermal_expansion = <real> (0)
  stress_free_temperature = <real>
[../]
```

type	CreepU10Mo

79

disp_x	Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.
disp_z	Variable name for displacement variable in z direction. Typically disp_z for 3D and disp_y for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically disp_x.
temp	Name of temperature variable. Typically temp.
fission_rate	Variable name corresponding to the fast neutron flux. Typically fission_rate.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
thermal_expansion	Coefficient of thermal expansion.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.

16.2.5 CreepUO2

The CreepUO2 is used to model the creep behavior of UO_2.

```
[./creepuo2]
  type = CreepUO2
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  burnup_function = <string>
  fission_rate = <string>
  density = <real>
  youngs_modulus = <real>
  poissons_ratio = <real>
  thermal_expansion = <real> (0)
  grain_radius = <real> (10e-6)
  oxy_to_metal_ratio = <real> (2)
  relative_tolerance = <real> (1e-4)
  absolute_tolerance = <real> (1e-20)
  max_its = <integer> (10)
  output_iteration_info = <true or false> (false)
  stress_free_temperature = <real>
  matpro_youngs_modulus = <true or false> (false)
  matpro_poissons_ratio = <true or false> (false)
  matpro_thermal_expansion = <true or false> (false)
```

```
   burnup = <string>
   isotropic_cracking = <true or false> (false)
   rod_ave_lin_pow = <string>
 [../]
```

type	CreepUO2
disp_x	Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.
disp_z	Variable name for displacement variable in z direction. Typically disp_z for 3D and disp_y for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically disp_x.
temp	Name of temperature variable. Typically temp.
burnup_function	Name of the Burnup sub-block, if any (typically burnup). May not be used with fission_rate. Use of burnup_function is preferred.
fission_rate	Variable name corresponding to the fission rate. Typically fission_rate.
density	**Required**. The initial fuel density.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
thermal_expansion	Coefficient of thermal expansion.
grain_radius	Fuel grain radius.
oxy_to_metal_ratio	Oxygen to metal ratio.
relative_tolerance	Relative convergence tolerance for material model iterations.
absolute_tolerance	Absolute convergence tolerance for material model iterations.
max_its	Maximum number of material model convergence iterations.
output_iteration_info	Whether to output material model convergence information.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.
matpro_youngs_modulus	Set to true to use correlations for Young's modulus from MATPRO [9].
matpro_poissons_ratio	Set to true to use correlations for Poisson's modulus from MATPRO [9].
matpro_thermal_expansion	Set to true to use correlations for coefficient of thermal expansion from MATPRO [9].

burnup	Name of burnup variable. Only required if using MATPRO correlations. Typically `burnup`.
isotropic_cracking	Set to true to use isotropic cracking model.
rod_ave_lin_pow	Name of linear power function. Only required if `isotropic_cracking = true`.

16.2.6 HotPressingUO2

```
[./hotpressinguo2]
  type = HotPressingUO2
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  fission_rate = <string>
  youngs_modulus = <real>
  poissons_ratio = <real>
  thermal_expansion = <real> (0)
  grain_radius = <real> (10e-6)
  oxy_to_metal_ratio = <real> (2)
  relative_tolerance = <real> (1e-4)
  absolute_tolerance = <real> (1e-20)
  max_its = <integer> (10)
  output_iteration_info = <true or false> (false)
  stress_free_temperature = <real>
  matpro_youngs_modulus = <true or false> (false)
  matpro_poissons_ratio = <true or false> (false)
  matpro_thermal_expansion = <true or false> (false)
  burnup = <string>
  abs_error = <real> (1.0e-6)
  compute_pressure = <true or false> (true)
  debug_output = <true or false> (false)
  has_hotpressing_plasticity  = <true or false> (false)
  hotpressing_alpha = <real> (0)
  hotpressing_nu = <real> (0)
  input_hardening_modulus = <real> (0)
  input_hotpressing = <true or false> (false)
  input_yield_stress = <real> (0)
  max_iteration = <integer> (1000)
  model_hotpressing = <true or false> (true)
  rel_error = <real> (0.001)
  use_flow_rule = <true or false> (true)
 [../]
```

type	HotPressingUO2

disp_x	Variable name for displacement variable in x direction. Typically `disp_x`.
disp_y	Variable name for displacement variable in y direction. Typically `disp_y`.
disp_z	Variable name for displacement variable in z direction. Typically `disp_z` for 3D and `disp_y` for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically `disp_x`.
temp	Name of temperature variable. Typically `temp`.
fission_rate	Variable name corresponding to the fission rate. Typically `fission_rate`.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
thermal_expansion	Coefficient of thermal expansion.
grain_radius	Fuel grain radius.
oxy_to_metal_ratio	Oxygen to metal ratio.
relative_tolerance	Relative convergence tolerance for material model iterations.
absolute_tolerance	Absolute convergence tolerance for material model iterations.
max_its	Maximum number of material model convergence iterations.
output_iteration_info	Whether to output material model convergence information.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.
matpro_youngs_modulus	Set to true to use correlations for Young's modulus from MATPRO [9].
matpro_poissons_ratio	Set to true to use correlations for Poisson's modulus from MATPRO [9].
matpro_thermal_expansion	Set to true to use correlations for coefficient of thermal expansion from MATPRO [9].
burnup	Name of burnup variable. Only required if using MATPRO correlations. Typically `burnup`.
abs_error	Absolute error in the sub-newton iteration loop to compute trial pressure.
compute_pressure	Set to true to compute trial pressure.
debug_output	Set to true to output debug information.

has_hotpressing_plasticity	Set to true to turn on instantaneous plasticity model.
hotpressing_alpha	Parameter alpha in the hot-pressing model.
hotpressing_nu	Parameter nu in the hot-pressing model.
input_hardening_modulus	Hardening modulus in a linear hardening model for UO_2.
input_hotpressing	Set true to input hot-pressing parameters.
input_yield_stress	Yield stress of UO_2.
max_iteration	Maximum iteration number in the sub-newton interation loop to compute trial pressure.
model_hotpressing	Set true to model hot-pressing of UO_2.
rel_error	Relative error in the sub-newton interation loop to compute trial pressure.
use_flow_rule	Use modified flow rule for modeling hot-pressing plasticity.

The HotPressingUO2 is used to model the stress induced densification of UO_2.

16.2.7 CreepUPuZr

CreepUPuZr is used to model the thermal and irradiation creep behavior of U-Pu-Zr fast reactor fuel. If porosity and porosity_material are both defined an error will result. The user has the choice of not supplying any porosity, supplying a variable for porosity or couple to a material that calculates the porosity.

CreepUPuZr can also be used to calculate the open pore compression strain increment (hot pressing strain), due to the closing of open pores. The hydrostatic_stress must be set to include the hot pressing strain in the model, and the plenum_pressure should also be defined to correctly calculate the hot pressing strain.

```
[./creepupuzr]
  type = CreepUPuZr
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  porosity = <string>
  porosity_material = <bool> (false)
  fission_rate = <string>
  youngs_modulus = <real>
  poissons_ratio = <real>
  hydrostatic_stress = <string>
  plenum_pressure = <string>
  gamma_transition = <real> (923)
```

84

```
    thermal_expansion = <real> (0)
    stress_free_temperature = <real>
  [../]
```

type	CreepUPuZr
disp_x	Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.
disp_z	Variable name for displacement variable in z direction. Typically disp_z for 3D and disp_y for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically disp_x.
temp	Name of temperature variable. Typically temp.
porosity	Variable name corresponding to the porosity. Typically porosity.
porosity_material	A flag indicating whether porosity is being calculated by a material property (e.g., VSwellingUPuZr). Optional.
fission_rate	Variable name corresponding to the fast neutron flux. Typically fission_rate.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
hydrostatic_stress	Variable name for the hydrostatic (mean) stress value. Typically hydrostatic_stress.
plenum_pressure	The postprocessor name for the plenum pressure. Typically plenum_pressure.
gamma_transition	Gamma phase transition temperature, in Kelvin.
thermal_expansion	Coefficient of thermal expansion.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.

16.2.8 Elastic

The Elastic model is a simple hypo-elastic model.

```
  [./elastic]
    type = Elastic
    disp_x = <string>
    disp_y = <string>
    disp_z = <string>
    disp_r = <string>
```

```
    temp = <string>
    youngs_modulus = <real>
    poissons_ratio = <real>
    thermal_expansion = <real> (0)
    stress_free_temperature = <real>
  [../]
```

type	Elastic
disp_x	Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.
disp_z	Variable name for displacement variable in z direction. Typically disp_z for 3D and disp_y for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically disp_x.
temp	Name of temperature variable. Typically temp.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
thermal_expansion	Coefficient of thermal expansion.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.

16.2.9 FeCrAlPlasticity

The IsoPlasticityFeCrAl model is a model used to model the instantaneous isotropic plasticity of FeCrAl cladding. A yield stress function as a function of temperature is automatically set based upon data in the literature. See the Theory Manual for details.

```
[./fecral_plasticity]
   type = IsoPlasticityFeCrAl
   relative_tolerance = <real> (1e-5)
   absolute_tolerance = <real> (1e-20)
   max_its = <integer> (30)
   output_iteration_info = <bool> (false)
   output_iteration_info_on_error = <bool> (false)
   hardening_constant = <real>
   hardening_function = <string>
   yield_stress = <real>
  [../]
```

type	IsoPlasticityFeCrAl

relative_tolerance	Relative convergence tolerance for sub-newton iterations.
absolute_tolerance	Absolute convergence tolerance for sub-newton iterations.
max_its	Maximum number of sub-newton convergence iterations.
output_iteration_info	Whether to output sub-newton convergence information.
output_iteration_info_on_error	Whether to output material model convergence information if an error occurs.
hardening_constant	Input hardening slope.
hardening_function	True stress as a function of plastic strain.
yield_stress	Input initial yield stress. This is overwritten by the yield stress as a function of temperature inherent to the model.

16.2.10 FailureFeCrAl

FailureFeCrAl models the failure of FeCrAl cladding. The hoop stress is compared against the ultimate tensile strength (UTS) which is determined as a function of temperature based upon data in the literature. See the Theory Manual for details.

```
[./failure_fecral]
   type = FailureFeCrAl
   block = <string list>
   boundary = <string list>
   temperature = <string>
   hoop_stress = <string>
[../]
```

type	FailureFeCrAl
block	**Required**. Only required if boundary not supplied. Block ids for which failure of elements will be calculated. Material property failed becomes 1.0 when failure of that element has occurred.
boundary	**Required**. Only required if block not supplied. Boundary ids for which failure of side surfaces will be calculated. Material property failed.
temp	**Required**. Name of temperature variable. Typically temp.
hoop_stress	**Required**. Name of the hoop stress aux variable. Typically hoop_stress.

16.2.11 FailureCladding

FailureCladding is the model for Zircaloy-4 cladding failure due to burst during a LOCA accident.

```
[./failurecladding]
  type = FailureCladding
  boundary = <string list>
  criterion = <integer> (2)
  hoop_stress = <string>
  eff_strain_rate_creep = <string>
  eff_strain_rate_plast = <string>
  temperature = temperature
  fract_beta_phase = <string>
  fract_oxygen_gain = <string>
[../]
```

type	FailureCladding
boundary	**Required**. Waterside cladding boundary. The model in the current version is applied to the boundary because it is coupled to the waterside oxidation model (Section 16.2.19).
criterion	Choice of the failure criterion. One of 0 (overstress), 1 (plastic instability) or 2 (combined overstress and plastic instability). `criterion = 2` is recommended.
hoop_stress	Name of hoop stress auxiliary variable. For evaluating overstress.
eff_strain_rate_creep	Name of the auxiliary variable for the effective strain rate due to creep. For evaluating plastic instability.
eff_strain_rate_plast	Name of the auxiliary variable for the effective strain rate due to plasticity. For evaluating plastic instability.
temperature	Name of temperature variable. For evaluating overstress.
fract_beta_phase	Name of the auxiliary variable for the volume fraction of Zr β phase, calculated by the model ZrPhase (Section 16.2.28).
fract_oxygen_gain	Name of the auxiliary variable for the gained weight fraction of oxygen in the cladding, calculated by the model Oxidation-Cladding (Section 16.2.19).

16.2.12 FailureCladHT9

`FailureCladHT9` is the model for HT9 cladding failure during both long and short transients. Long transient failure is traditionally predicted with the Cumulative Damage Fraction (CDF) method. Short transients track cavity growth along grain boundaries with a Constrained Cavity Growth (CCG) with Diffusion and Creep with Sliding (D&CS) mechanism.

```
[./failurecladht9]
  type = FailureCladHT9
  block = <string list>
  boundary = <string list>
```

```
   transient = <boolean>
   temperature = temperature
   hoop_stress = hoop_stress
   boltzmann = <real> (1.3806488e-23)
   q_val = <real> (6.44336e5)
   avogadro_num = <real> (6.0221409e23)
   a_initial = <real> (7.0e-08)
   atomic_volume = <real> (1.18e-29)
   b = <real> (3.5e-06)
   boundary_diffusivity = <real> (1.1e-12)
   boundary_free_energy = <real> (0.85)
   creep_n_power = <real> (5.0)
   eff_strain_rate_creep = <string>
   hydrostatic_stress = <string>
   surface_free_energy = <real> (2.1)
   von_mises_stress = <string>
 [../]
```

type	FailureCladHT9
block	**Required**. Only required if `boundary` not supplied. Block ids for which failure of elements will be calculated. Material property `failed` becomes 1.0 when failure of that element has occurred.
boundary	**Required**. Only required if `block` not supplied. Boundary ids for which failure of side surfaces will be calculated. Material property `failed` becomes 1.0 when failure of that side surface has occurred.
transient	**Required**. CCG with D&CS activated with `true`. CDF activated with `false`.
temperature	**Required**. Name of temperature variable.
hoop_stress	**Required**. Name of variable containing the hoop stress.
boltzmann	Boltzmann constant. Used for both CCG and CDF.
q_val	*CDF*. Q value for data collapse to the Dorn parameter.
avogadro_num	*CDF*. Avogadro's number.
a_initial	*CCG*. Initial cavity radius. Cannot be equal to zero.
atomic_volume	*CCG*. Atomic volume of the material.
b	*CCG*. Distance from cavity center to midplane between cavities. 2*b* is the distance between periodic cavity centers.
boundary_diffusivity	*CCG*. Grain boundary diffusivity.
boundary_free_energy	*CCG*. Grain boundary free energy.
creep_n_power	*CCG*. Exponent power *n* for a material obeying a creep power law.
eff_strain_rate_creep	*CCG*. Variable containing the effective creep strain rate value.

hydrostatic_stress	*CCG*. Variable containing the hydrostatic (mean) stress value.
surface_free_energy	*CCG*. Grain surface free energy.
von_mises_stress	*CCG*. Variable containing the Von Mises stress value.

16.2.13 IrradiationGrowthZr4

The IrradiationGrowthZr4 model incorporates anisotropic volumetric swelling to track axial elongation in Zr4 cladding.

```
[./irradiationgrowthzr4]
  type = IrradiationGrowthZr4
  fast_neutron_fluence = <string>
  Ag = <real> (3e-20)
  ng = <real> (0.794)
[../]
```

type	IrradiationGrowthZr4
fast_neutron_fluence	Name of fast neutron fluence variable. Typically fast_neutron_fluence.
Ag	Material constant that depends on the cladding metalurgical state.
ng	Material constant that depends on the cladding metalurgical state.

16.2.14 MechFeCrAl

The MechFeCrAl model computes the elastic moduli, coefficient of thermal expansion, and thermal creep of a variety of FeCrAl alloys being considered for accident tolerant cladding.

```
[./mechFeCrAl]
  type = MechFeCrAl
  block = <string list>
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  fast_neutron_flux = <string>
  youngs_modulus = <real>
  poissons_ratio = <real>
  material = <string> (APMT)
  scale_factor_A = <real> (1.0)
  scale_factor_Q = <real> (1.0)
  scale_factor_n = <real> (1.0)
  scale_factor_alpha = <real> (1.0)
  scale_factor_cte = <real> (1.0)
  scale_factor_youngs = <real> (1.0)
  scale_factor_nu = <real> (1.0)
```

```
   model_thermal_expansion = <bool> (true)
   model_elastic_modulus = <bool> (false)
   model_thermal_creep = <bool> (true)
   model_irradiation_creep = <bool> (true)
   stress_free_temperature = <real>
 [../]
```

type	MechFeCrAl.
block	The list of blocks this material applies to.
disp_x	Variable name for displacement variable in x direction. Typically `disp_x`.
disp_y	Variable name for displacement variable in y direction. Typically `disp_y`.
disp_z	Variable name for displacement variable in z direction. Typically `disp_z` for 3D and `disp_y` for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically `disp_x`.
temp	Name of temperature variable. Typically `temp`.
fast_neutron_flux	Name of the fast flux variable. Typically `fast_neutron_flux`.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
material	Name of chosen FeCrAl alloy. Choices are MA956, PM2000, APMT, FECRALLOY, C35M.
scale_factor_A	Scale factor to be applied to the creep prefactor.
scale_factor_Q	Scale factor to be applied to the creep activation energy.
scale_factor_n	Scale factor to be applied to the creep stress exponent.
scale_factor_alpha	Scale factor to be applied to the temperature coefficient.
scale_factor_cte	Scale factor to be applied to the thermal expansion coefficient.
scale_factor_youngs	Scale factor to be applied to the Young's modulus.
scale_factor_nu	Scale factor to be applied to the Poisson's ratio.
model_thermal_expansion	Whether to calculate the thermal expansion coefficient as a function of temperature.
model_elastic_modulus	Whether to calculate temperature dependent elastic moduli.
model_thermal_creep	Whether or not to model thermal creep.
model_irradiation_creep	Whether or not to model irradiation creep.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.

16.2.14.1 MechFeCrAlModel

The `MechFeCrAlModel` model is the default constitutive model used in MechFeCrAl. This model is used as a separate block in the input file when it is specified as one of the submodels in `CombinedCreepPlasticity` model.

```
[./MechFeCrAlModel]
    type = MechFeCrAlModel
    material = <string> (APMT)
    temp = <string>
    fast_neutron_flux = <string>
    scale_factor_A = <real> (1.0)
    scale_factor_Q = <real> (1.0)
    scale_factor_n = <real> (1.0)
    scale_factor_alpha = <real> (1.0)
    scale_factor_cte = <real> (1.0)
    scale_factor_irrad = <real> (1.0)
    creep_coefficient = <real> (2.89e-36)
    stress_exponent = <real> (5.5)
    activation_energy = <real> (29709)
    model_thermal_expansion = <bool> (true)
    model_thermal_creep     = <bool> (true)
    model_irradiation_creep = <bool> (true)
    relative_tolerance = <real> (1e-5)
    absolute_tolerance = <real> (1e-20)
    max_its = <integer> (10)
    output_iteration_info = <bool> (false)
    output_iteration_info_on_error = <bool> (false)
    max_creep_increment = <real> (1.e-03)
    stress_free_temperature = <real>
[../]
```

type	MechFeCrAlModel
material	Name of chosen FeCrAl alloy. Choices are MA956, PM2000, APMT, FECRALLOY, C35M.
temp	Name of temperature variable. Typically `temp`.
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically `fast_neutron_flux`.
scale_factor_A	Scale factor to be applied to the creep prefactor.
scale_factor_Q	Scale factor to be applied to the creep activation energy.
scale_factor_n	Scale factor to be applied to the creep stress exponent.
scale_factor_alpha	Scale factor to be applied to the temperature coefficient.
scale_factor_cte	Scale factor to be applied to the thermal expansion coefficient.

scale_factor_irrad	Scale factor to be applied to the irradiation creep.
model_thermal_creep	Whether to model thermal creep.
creep_coefficient	Pre–exponential coefficient in the thermal creep correlation. The default value is for C35M.
stress_exponent	Exponent applied to the stress in the thermal creep correlation. The default value is for C35M.
activation_energy	Activation energy for thermal creep. The default value is for C35M.
model_irradiation_creep	Whether to model irradiation-induced creep.
model_thermal_expansion	Whether to calculate the thermal expansion coefficient as a function of temperature.
relative_tolerance	Relative convergence tolerance for material model iterations.
absolute_tolerance	Absolute convergence tolerance for material model iterations.
max_its	Maximum number of material model convergence iterations.
output_iteration_info	Whether to output material model convergence information.
output_iteration_info_on_error	Whether to output material model convergence information if an error occurs.
max_creep_increment	The maximum increment of effective creep strain allowed by the time step control.

16.2.15 MechMaterial

The MechMaterial model computes the elastic moduli and thermal expansion a variety of materials. The MechMaterial is used to describe a variety of materials that have the same form in the input file. The choices are MechAlloy33, MechHT9, MechMo, MechSS316 and MechU3Si5UN. These materials are typically used as cladding materials. Examples of their use can be found in /bison/tests/HT9 and /bison/tests/mechTests/.

```
[./mechMaterial]
  type = Mech<string>
  block = <string list>
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  youngs_modulus = <real>
  poissons_ratio = <real>
[../]
```

type	Mech<string>. Where <string> represents the particular material to be used (e.g. HT9).
block	The list of blocks this material applies to.
disp_x	Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.
disp_z	Variable name for displacement variable in z direction. Typically disp_z for 3D and disp_y for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically disp_x.
temp	Name of temperature variable. Typically temp.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.

16.2.16 MechMAMOX

The MechMAMOX model computes the thermal expansion of minor actinide doped mixed oxide fast fuel (MA-MOX). The Young's modulus and Poisson's ratios are set default to 249.3e9 Pa and 0.317, respectively, but can be changed in the input file if desired. This model was built based on the correlation found here [10] and is good for oxygen to metal ratios of 2.00, 1.99, 1.98 and 1.97. Oxygen to metal ratio is a required parameter. The correlation describes a mean thermal expansion coefficient with a reference temperature of 300 K. porosity can either be coupled or defined as a function in the input file.

```
[./mechMAMOX]
  type = MechMAMOX
  block = <string list >
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  oxy_to_metal_ratio = <string> (required)
  porosity = <string>
  youngs_modulus = <real>
  poissons_ratio = <real>
[../]
```

type	MechMAMOX.
block	The list of blocks this material applies to.
disp_x	Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.

disp_z	Variable name for displacement variable in z direction. Typically disp_z for 3D and disp_y for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically disp_x.
temp	Name of temperature variable. Typically temp.
oxy_to_metal_ratio	The oxygen to metal ratio of the fuel, this can be 2.00, 1.99, 1.98 or 1.97 only and will error with anything else.
porosity	Variable name corresponding to the porosity. Typically porosity.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.

16.2.17 MechUPuZr

MechUPuZr is used to calculate mechanical properties and thermal expansion, as well to model thermal and irradiation creep behavior of U-Pu-Zr fast reactor fuel. If porosity and porosity_material are both defined an error will result. The user has the choice of not supplying any porosity, supplying a variable for porosity or couple to a material that calculates the porosity.

```
[./mechupuzr]
  type = MechUPuZr
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  X_Pu = <string>
  X_Zr = <string>
  porosity = <string>
  porosity_material = <bool> (false)
  fission_rate = <string>
  youngs_modulus = <real>
  poissons_ratio = <real>
  thermal_expansion = <real> (0)
  stress_free_temperature = <real>
  model_creep = <bool> (true)
  calc_youngs = <bool> (true)
  calc_poissons = <bool> (true)
  model_swelling = <bool> (false)
  name_swelling_model = <string> (VSwellingUPuZr)
  LTE_scalar = <real> (1.0)
  A_U = <real> (0.2380289)
  A_Pu = <real> (0.244)
  A_Zr = <real> (0.091224)
[../]
```

type   MechUPuZr

95

disp_x	Variable name for displacement variable in x direction. Typically `disp_x`.
disp_y	Variable name for displacement variable in y direction. Typically `disp_y`.
disp_z	Variable name for displacement variable in z direction. Typically `disp_z` for 3D and `disp_y` for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically `disp_x`.
temp	Name of temperature variable. Typically `temp`.
X_Pu	Name of plutonium molar fraction variable. Typically `X_Pu`.
X_Zr	Name of zirconium molar fraction variable. Typically `X_Zr`.
porosity	Variable name corresponding to the porosity. Typically `porosity`.
porosity_material	A flag indicating whether porosity is being calculated by a material property (e.g., VSwellingUPuZr). Optional.
fission_rate	Variable name corresponding to the fast neutron flux. Typically `fission_rate`.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
thermal_expansion	Coefficient of thermal expansion.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.
model_creep	Flag to model creep using CreepUPuZrModel.
calc_youngs	Flag to calculate Young's Modulus as a function of molar fractions, temperature, and porosity.
calc_poissons	Flag to calculate Poisson's Ratio as a function of molar fractions, and temperature.
model_swelling	Flag to model swelling.
name_swelling_model	Material name for swelling model.
LTE_scalar	Multiplier multiplied against the calculated linear expansion coefficient. Use 0 to turn thermal expansion off.
A_U	Uranium molar mass.
A_Pu	Plutonium molar mass.
A_Zr	Zirconium molar mass.

16.2.18 MechZry

The `MechZry` model includes the options to model primary, thermal, and irradiation-induced creep. Modeling of creep under LOCA conditions is available. Also, it is possible to turn on irradiation growth. If irradiation growth is turned on, do not include the `IrradiationGrowthZr4` model. Finally, an accuracy-controlling time step criterion is implemented, which can be applied through the `MaterialTimeStep` postprocessor (Section 18.7).

```
[./mechzry]
  type = MechZry
  fast_neutron_flux = <string>
  fast_neutron_fluence = <string>
  initial_fast_fluence = <real> (0.0)
  cold_work_factor = <real> (0.01)
  constitutive_model = <string>
  oxygen_concentration = <real> (0.0)
  relative_tolerance = <real> (1e-4)
  absolute_tolerance = <real> (1e-20)
  max_its = <integer> (10)
  output_iteration_info = <bool> (false)
  output_iteration_info_on_error = <bool> (false)
  model_irradiation_creep = <bool> (true)
  model_primary_creep = <bool> (true)
  model_thermal_creep = <bool> (true)
  model_thermal_creep_loca = <bool> (false)
  model_irradiation_growth = <bool> (true)
  growth_direction = <integer>
  model_thermal_expansion = <bool> (true)
  model_elastic_modulus = <bool> (false)
  stress_free_temperature = <real>
  material_type = <string>(SRA) or <integer>(0)
  temp_standard_creep_end = <real> (700.)
  temp_loca_creep_begin = <real> (900.)
  max_creep_increment = <real> (1.e-03)
  creeprate_scalef = <real> (1.)
  constitutive_model = <string>
  scale_factor = <real> (1.0)
[../]
```

type	MechZry
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.
fast_neutron_fluence	Name of fast neutron fluence variable. Typically fast_neutron_fluence.
initial_fast_fluence	The initial fast neutron fluence.
constitutive_model	Name of constitutive model.
cold_work_factor	Cold work factor.

oxygen_concentration	Oxygen concentration in ppm.
relative_tolerance	Relative convergence tolerance for material model iterations.
absolute_tolerance	Absolute convergence tolerance for material model iterations.
max_its	Maximum number of material model convergence iterations.
output_iteration_info	Whether to output material model convergence information.
output_iteration_info_on_error	Whether to output material model convergence information if an error occurs.
model_irradiation_creep	Whether to model irradiation-induced creep.
model_primary_creep	Whether to model primary creep.
model_thermal_creep	Whether to model steady state thermal creep.
model_thermal_creep_loca	Whether to model thermal creep considering a specific correlation at LOCA temperatures.
model_irradiation_growth	Whether to model irradiation growth.
growth_direction	The direction in which irradiation growth is applied. Typically 1 (axial direction).
model_thermal_expansion	Whether to use MATPRO model for thermal expansion.
model_elastic_modulus	Whether to calculate temperature-dependent elastic moduli.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.
material_type	Cladding material type. SRA(or 0), RXA(or 1), PRXA(or 2), ZIRLO(or 3).
temp_standard_creep_end	The upper limit of temperature where the standard thermal creep model no longer applies.
temp_loca_creep_begin	The lower limit of temperature where the loca thermal creep model begins to apply.
max_creep_increment	The maximum increment of effective creep strain allowed by the time step control.
creeprate_scalef	Scale factor applied to the creep rate (for sensitivity studies).
constitutive_model	By default (when the constitutive model parameter is not given in input file) CreepZryModel constitutive model is used else the constitutive model specified by the user is used.

scale_factor	Young's modulus calculated by the MATPRO CELMOD function is scaled down by this value.

16.2.18.1 CombinedCreepPlasticity

The `CombinedCreepPlasticity` model is used when it is specified as the `constitutive_model` in `MechZry`. `CombintedCreepPlasticity` models the deformation under both creep and instanteous plasticity. Submodels need to be specified in the input to model creep and instantaneous plasticity of Zry cladding.

```
[./CombinedCreepPlasticity]
    type = CombinedCreepPlasticity
    relative_tolerance = <real> (1e-5)
    absolute_tolerance = <real> (1e-5)
    max_its = <integer> (30)
    output_iteration_info = <bool> (false)
    output_iteration_info_on_error = <bool> (false)
    submodels = <string>
[../]
```

type	CombinedCreepPlasticity
relative_tolerance	Relative convergence tolerance for combined submodel iterations.
absolute_tolerance	Absolute convergence tolerance for combined submodel iterations.
max_its	Maximum number of submodel convergence iterations.
output_iteration_info	Whether to output submodel convergence information.
output_iteration_info_on_error	Whether to output submodel convergence information if an error occurs.
submodels	Name of submodels used for combined creep and plasticity.

16.2.18.2 CreepZryModel

The `CreepZryModel` model is the default constitutive model used in MechZry. Therefore, the interface is similar to the `MechZry` except for the options and paramters to model elastic modulus, thermal expansion, and irradiation growth. This model is used as a separate block in the input file when it is specified as one of the submodels in `CombinedCreepPlasticity` model.

```
[./CreepZryModel]
    type = CreepZryModel
    fast_neutron_flux = <string>
    fast_neutron_fluence = <string>
```

```
   initial_fast_fluence = <real>
   cold_work_factor = <real> (0.01)
   oxygen_concentration = <real> (0.0)
   relative_tolerance = <real> (1e-5)
   absolute_tolerance = <real> (1e-20)
   max_its = <integer> (30)
   output_iteration_info = <bool> (false)
   output_iteration_info_on_error = <bool> (false)
   model_irradiation_creep = <bool> (true)
   model_primary_creep = <bool> (true)
   model_thermal_creep = <bool> (true)
   model_thermal_creep_loca = <bool> (false)
   material_type = <string>(SRA) or <integer>(0)
   temp_standard_creep_end = <real> (700.)
   temp_loca_creep_begin = <real> (900.)
   max_creep_increment = <real> (1.e-03)
   creeprate_scalef = <real> (1.)
[../]
```

type	CreepZryModel
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.
fast_neutron_fluence	Name of fast neutron fluence variable. Typically fast_neutron_fluence.
initial_fast_fluence	The initial fast neutron fluence.
cold_work_factor	Cold work factor.
oxygen_concentration	Oxygen concentration in ppm.
relative_tolerance	Relative convergence tolerance for material model iterations.
absolute_tolerance	Absolute convergence tolerance for material model iterations.
max_its	Maximum number of material model convergence iterations.
output_iteration_info	Whether to output material model convergence information.
output_iteration_info_on_error	Whether to output material model convergence information if an error occurs.
model_irradiation_creep	Whether to model irradiation-induced creep.
model_primary_creep	Whether to model primary creep.
model_thermal_creep	Whether to model steady state thermal creep.
model_thermal_creep_loca	Whether to model thermal creep considering a specific correlation at LOCA temperatures.

material_type	Cladding material type. SRA(or 0), RXA(or 1), PRXA(or 2), or ZIRLO(or 3).
temp_standard_creep_end	The upper limit of temperature where the standard thermal creep model no longer applies.
temp_loca_creep_begin	The lower limit of temperature where the loca thermal creep model begins to apply.
max_creep_increment	The maximum increment of effective creep strain allowed by the time step control.
creeprate_scalef	Scale factor applied to the creep rate (for sensitivity studies).

16.2.18.3 IsotropicPlasticity

The IsotropicPlasticity model is a model used to model the instantaneous plasticity of Zry cladding. It is used as one of the submodels in CombinedCreepPlasticity model.

```
[./IsotropicPlasticity]
   type = IsotropicPlasticity
   relative_tolerance = <real> (1e-5)
   absolute_tolerance = <real> (1e-20)
   max_its = <integer> (30)
   output_iteration_info = <bool> (false)
   output_iteration_info_on_error = <bool> (false)
   hardening_constant = <real>
   hardening_function = <string>
   yield_stress = <real>
 [../]
```

type	IsotropicPlasticity
relative_tolerance	Relative convergence tolerance for sub-newton iterations.
absolute_tolerance	Absolute convergence tolerance for sub-newton iterations.
max_its	Maximum number of sub-newton convergence iterations.
output_iteration_info	Whether to output sub-newton convergence information.
output_iteration_info_on_error	Whether to output material model convergence information if an error occurs.
hardening_constant	Input hardening slope.
hardening_function	True stress as a function of plastic strain.
yield_stress	Input initial yeild stress.

16.2.18.4 ZryPlasticity

The `ZryPlasticity` model is used to model the instantaneous plasticity of the Zry cladding. The stress vs total strain curve after yield follows a power law hardening. The yield stress and hardening curve is calculated based on the fast neutron fluence, fast neutron flux, temperature and cold work factor. This model is used in conjunction with MechZry. The Young's modulus and Poisson's ratio are evaluated by MechZry (using the MATPRO CELMOD subroutine) as a function of fast neutron fluence, temperature, cold work factor and oxygen concentration. So model_elastic_modulus in MechZry model should be set to true and all the other modeling parameters like model_thermal_expansion, model_primary_creep, e.t.c, should be set to false.

A scale factor can be applied to scale down the Young's modulus, yield stress and hardening modulus. The yield stress and hardening modulus are scaled in ZryPlasticity and the Young's modulus is scaled in MechZry.

```
[./ZryPlasticity]
  type = ZryPlasticity
  relative_tolerance = <real> (1e-5)
  absolute_tolerance = <real> (1e-20)
  max_its = <integer> (30)
  output_iteration_info = <bool> (false)
  output_iteration_info_on_error = <bool> (false)
  fast_neutron_flux = <string>
  fast_neutron_fluence = <string>
  temp = <string>
  cold_work_factor = <real> (0.0)
  Ziracloy_type = <unsigned int> (4)
  use_matpro = <bool> (false)
  scale_factor = <real> (1.0)
  initial_fast_fluence = <real> (0.0)
  strain_rate = <real>
[../]
```

type	ZryPlasticity
relative_tolerance	Relative convergence tolerance for sub-newton iterations.
absolute_tolerance	Absolute convergence tolerance for sub-newton iterations.
max_its	Maximum number of sub-newton convergence iterations.
output_iteration_info	Whether to output sub-newton convergence information.
output_iteration_info_on_error	Whether to output material model convergence information if an error occurs.
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.

fast_neutron_fluence	Name of fast neutron fluence variable. Typically fast_neutron_fluence.
temp	Name of the temperature variable.
cold_work_factor	Cold work factor.
Ziracloy_type	Type of Ziracloy used - 2 or 4.
use_matpro	Parameters for the power law hardening model (K.n and m) will be modeled using the expression in MAT-PRO if this is set to true. A fixed strain rate of 1e-3 is used with this model.
scale_factor	Yield stress and Hardening modulus are scaled down by this parameter.
initial_fast_fluence	The initial value of fast fluence variable.
strain_rate	Fixes strain rate to the value provided. Works only with the default model (use_matpro = false). When use_matpro is set to true, the strain rate is automatically set to 1e-3.

16.2.19 OxidationCladding

The `OxidationCladding` model incorporates correlations for Zircaloy cladding oxidation through metal-water reactions. Described processes include outer oxide scale thickness growth and oxygen mass gain. The model is to be applied to the cladding waterside boundary.

```
[./oxidationcladding]
  type = OxidationCladding
  boundary = <string list>
  clad_inner_radius = <real>
  clad_outer_radius = <real>
  normtemp_model = 0
  hightemp_model = <integer> (0)
  temperature = <string>
  fast_neutron_flux = <string>
  use_coolant_channel = <bool> (false)
  oxygen_wtfract_initial = <real> (0.0012)
  oxidation_scalef = <real> (1.)
[../]
```

type	OxidationCladding
boundary	**Required**. Waterside cladding boundary.
clad_inner_radius	**Required**. Inner cladding radius (m).
clad_outer_radius	**Required**. Outer cladding radius (m).
normtemp_model	Selection of the normal operating temperatures correlation. Only 0 (EPRI/KWU/C-E) is considered at this time.

103

hightemp_model	Selection of the high temperatures correlation. One of 0 (Leis-tikov) or 1 (Cathcart-Pawel). In both cases, the model uses the Prater-Courtright correlation above 1900 K.
temp	**Required**. Name of temperature variable. Typically temp.
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.
use_coolant_channel	If true, the model will adjust the surface temperature based on the coolant channel model.
oxygen_wtfract_initial	As-fabricated oxygen weight fraction in cladding.
oxidation_scalef	Scale factor applied to the oxide scale thickness (for sensitivity studies).

16.2.20 RelocationUO2

The RelocationUO2 model accounts for cracking and relocation of fuel pellet fragments in the radial direction. This model is necessary for accurate modeling of LWR fuel. Only one of q and q_variable may be given.

```
[./relocationuo2]
  type = RelocationUO2
  burnup_function = <string>
  burnup = <string>
  fuel_pin_geometry = <string>
  diameter = <real>
  q = <string>
  q_variable = <string>
  gap = <real>
  burnup_relocation_stop = <real>
  relocation_activation1 = <real> (19685.039)
  relocation_activation2 = <real> (45931.759)
  relocation_activation3 = <real> (32808.399)
  axial_axis = <0, 1, or 2 for x, y, or z>
  model = <ESCORE_modified, ESCORE, or GAPCON> (ESCORE_modified)
 [../]
```

type	RelocationUO2
burnup_function	Name of the Burnup sub-block, if any (typically burnup). May not be used with the burnup line command. Use of burnup_function is preferred.
burnup	Name of burnup variable. Typically burnup.
fuel_pin_geometry	Name of the FuelPinGeometry object (see 23.1).
diameter	**Required if fuel_pin_geometry is not specified**. As fabricated cold diameter of pellet in meters.
q	Function describing linear heat rate in pellet in W/m.

q_variable	Variable holding linear heat rate in pellet in W/m.
gap	**Required if fuel_pin_geometry is not specified**. As fabricated cold diametral gap in m.
burnup_relocation_stop	Burnup at which relocation strain stops in FIMA.
relocation_activation1	First activation linear power in W/m. The linear power at which relocation turns on.
relocation_activation2	Second activation linear power in W/m. The linear power at which relocation transitions from the initial regime to the secondary regime.
relocation_activation3	Third activation linear power in W/m. The linear power offset in the secondary regime.
axial_axis	Coordinate axis of the axial direction of the fuel stack.
model	Which relocation correlation to use.

16.2.21 ThermalIrradiationCreepZr4

The `ThermalIrradiationCreepZr4` is used for Zr4 cladding in LWR simulations. It includes fits for the temperature, irradiation, and stress effects on cladding creep.

```
[./thermalirradiationcreepzr4]
  type = ThermalIrradiationCreepZr4
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  temp = <string>
  a_coeff = <real> (3.14e24)
  n_exponent = <real> (5)
  activation_energy = <real> (2.7e5)
  gas_constant = <real> (8.3143)
  fast_neutron_flux = <string>
  c0_coef = <real> (9.881e-28)
  c1_coef = <real> (0.85)
  c2_coef = <real> (1)
  youngs_modulus = <real>
  poissons_ratio = <real>
  thermal_expansion = <real> (0)
  relative_tolerance = <real> (1e-4)
  absolute_tolerance = <real> (1e-20)
  max_its = <integer> (10)
  output_iteration_info = <true or false> (false)
  stress_free_temperature = <real>
[../]
```

type	ThermalIrradiationCreepZr4

disp_x	Variable name for displacement variable in x direction. Typically disp_x.
disp_y	Variable name for displacement variable in y direction. Typically disp_y.
disp_z	Variable name for displacement variable in z direction. Typically disp_z for 3D and disp_y for axisymmetric models.
disp_r	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically disp_x.
temp	Name of temperature variable. Typically temp.
a_coef	The leading coefficient in the thermal creep term.
n_exponent	The exponent in the thermal creep term.
activation_energy	The activation energy.
gas_constant	The universal gas constant.
fast_neutron_flux	Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.
c0_coef	The leading coefficient in the irradiation creep term.
c1_exponent	The exponent on the irradiation creep fast neutron flux term.
c2_exponent	The exponent on the irradiation creep stress term.
youngs_modulus	Young's modulus.
poissons_ratio	Poisson's ratio.
thermal_expansion	Coefficient of thermal expansion.
relative_tolerance	Relative convergence tolerance for material model iterations.
absolute_tolerance	Absolute convergence tolerance for material model iterations.
max_its	Maximum number of material model convergence iterations.
output_iteration_info	Whether to output material model convergence information.
stress_free_temperature	The stress-free temperature. If not specified, the initial temperature is used.
burnup	Name of burnup variable. Typically burnup.

16.2.22 PyCIrradiationStrain

The PyCIrradiationStrain model tracks the irradiation-induced strain in pyrolytic carbon. The strain is isotropic for the buffer type and differs in the radial and tangential directions for the dense type.

```
[./pycirradiationstrain]
  type = PyCIrradiationStrain
  fluence = <string>
  pyc_type = <string> (buffer)
```

```
[../]
```

type	PyCIrradiationrStrain
fluence	**Required**. Variable name corresponding to the fast neutron fluence. Typically fast_neutron_fluence.
pyc_type	One of buffer or dense.

16.2.23 VSwellingFeCrAl

The VSwellingFeCrAl model computes a volumetric strain to account for irradiation induced swelling of FeCrAl alloys used for cladding.

```
[./vswelling_fecral]
  type = VSwellingFeCrAl
  fast_neutron_fluence = <string>
  swelling_scalef = <real> (1.0)
[../]
```

type	VSwellingFeCrAl
fast_neutron_fluence	The name of the variable for the fast neutron fluence.
swelling_scalef	A scaling factor to be applied to the swelling creep strain.

16.2.24 VSwellingUO2

The VSwellingUO2 model computes a volumetric strain to account for solid and gaseous swelling and for densification. VSwellingUO2 also computes porosity (fabrication_porosity, gaseous_porosity, and sintering_porosity).

```
[./vswellinguo2]
  type = VSwellingUO2
  temp = <string>
  burnup_function = <string>
  burnup = <string>
  density = <real>
  total_densification = <real> (0.01)
  complete_burnup = <real> (5)
  initial_porosity = <real> (0.05)
  save_solid_swell = <bool> (false)
  save_densification = <bool> (false)
  gas_swelling_type = <string> (SIFGRS)
[../]
```

type	VSwellingUO2
temp	Name of temperature variable. Typically temp.

burnup_function	Name of the `Burnup` sub-block, if any (typically `burnup`). May not be used with the `burnup` line command. Use of `burnup_function` is preferred.
burnup	Name of burnup variable. Typically `burnup`.
density	**Required**. Initial fuel density.
total_densification	The densification that will occur given as a fraction of theoretical density.
complete_burnup	The burnup at which densification is complete (MWd/kgU).
initial_porosity	The initial fuel porosity.
save_solid_swell	Whether to save the solid swelling as a material property (solid_swell).
save_densification	Whether to save the densification as a material property (densification).
gas_swelling_type	Either `SIFGRS` to use the gas swelling computed by the Sifgrs model or `MATPRO` to use MATPRO's empirical model. IMPORTANT: if using `SIFGRS`, the following line must be place in the block defining the solid mechanics material model for the fuel (e.g., `CreepUO2` or `Elastic`) so that the models run in the correct order: "dep_matl_props = deltav_v0_bd"

Note that the total swelling will be stored as a material property (named total_swell) if both save_solid_swell and save_densification are true. Solid swelling is saved as solid_swell, and densification is saved as densification. The total swelling includes solid swelling, densification, and gaseous swelling. Gaseous swelling is always stored as a material property (named gas_swell).

16.2.25 VSwellingU3Si2

The `VSwellingU3Si2` model computes a volumetric strain to account for solid and gaseous swelling and densification In U_3Si_2 fuel.

```
[./vswellingu3si2]
  type = VSwellingU3Si2
  temp = <string>
  burnup_function = <string>
  burnup = <string>
  total_densification = <real> (0.01)
  complete_burnup = <real> (5)
  save_solid_swell = <bool> (false)
  save_densification = <bool> (false)
  save_gas_swell = <bool> (false)
[../]
```

type	VSwellingU3Si2

temp	Name of temperature variable. Typically `temp`.
burnup_function	Name of the `Burnup` sub-block, if any (typically `burnup`). May not be used with the `burnup` line command. Use of `burnup_function` is preferred.
burnup	Name of burnup variable. Typically `burnup`.
total_densification	The densification that will occur given as a fraction of theoretical density.
complete_burnup	The burnup at which densification is complete (MWd/kgU).
save_solid_swell	Whether to save the solid swelling as a material property (solid_swelling).
save_densification	Whether to save the densification as a material property (densification).
save_gas_swell	Whether to save the gaseous swelling as a material property (gaseous_swelling).

Note that the total swelling will be stored as a material property (named total_swell) if `save_solid_swell`, `save_gas_swell`, and `save_densification` are true. Solid swelling is saved as solid_swelling, gaseous swelling is saved as gaseous_swelling, and densification is saved as densification. The total swelling includes solid swelling, densification, and gaseous swelling.

16.2.26 VSwellingU3Si5UN

The `VSwellingU3Si5UN` model computes a volumetric strain to account for solid and gaseous swelling and for densification.

```
[./vswellingU3Si5UN]
  type = VSwellingU3Si5UN
  temp = <string>
  burnup_function = <string>
  burnup = <string>
  density = <real>
  initial_porosity = <real> (0.05)
[../]
```

type	VSwellingU3Si5UN
temp	Name of temperature variable. Typically `temp`.
burnup_function	Name of the `Burnup` sub-block, if any (typically `burnup`). May not be used with the `burnup` line command. Use of `burnup_function` is preferred.
burnup	Name of burnup variable. Typically `burnup`.
density	**Required**. Initial fuel density.
initial_porosity	The initial fuel porosity.

16.2.27 VSwellingUPuZr

The `VSwellingUPuZr` model computes a volumetric strain to account for solid and gaseous swelling and for open pore compression in U-Pu-Zr metal fuel systems. The solid swelling and gaseous swelling are optionally saved as a material property, named solid_swell and gas_swell, respectively. Also, porosity (as-fabricated + gas swelling porosity) is available as a material property. The compressive strain increment due to open pore compression (hot pressing) is computed in `CreepUPuZrModel` and passed to `VSwellingUPuZr` as the material property open_pore_compression.

The dilatational components of the strain increment tensor can be scaled in this model with a user-defined input parameter called the `anisotropic_strain_scaling` vector. The default value of this parameter is '1 1 1', where these components are of type Real in C++ parlance. Those components can be changed, but are limited such that the sum of the components divided by 3 must equal 1 and the components can only be positive and less than or equal to 3. For example, taking the total amount of strain and applying it all in the first component of the strain increment tensor can be done by setting `anisotropic_strain_scaling` to '3 0 0'. This ensures that the total amount of strain is preserved.

To include the open core compression strain in `VSwellingUPuZr`, the `hydrostatic_stress` and `plenum_pressure` must be defined in the `CreepUPuZr` block.

```
[./vswelling_upuzr]
  type = VSwellingUPuZr
  anisotropic_strain_scaling = <three component vector> (1.0 1.0 1.0)
  temp = <string>
  fission_rate = <string>
  fabrication_porosity = <real> (0.0)
  hydrostatic_stress = <string>
  hot_pressing_strain_increment = <string>
  plenum_pressure = <string>
  save_gas_swell = <bool> (false)
  save_solid_swell = <bool> (false)
[../]
```

type	VSwellingUPuZr
anisotropic_strain_scaling	The scale factor applied to each component of the dilatational strain computed in this model. When defining these scale factors ensure that total strain is preserved.
temp	Name of temperature variable. Typically temp.
fission_rate	Name of fission rate variable. Typically fission_rate.
fabrication_porosity	The as-fabricated porosity

`hydrostatic_stress`	The name of hydrostatic stress variable. Typically `hydrostatic_stress`.
`hot_pressing_strain_increment`	The name of variable storing increment of strain due to compression of the open pores. Typically `open_pore_compression`
`plenum_pressure`	The postprocessor name for the plenum pressure. Typically `plenum_pressure`.
`save_gas_swell`	A flag indicating whether the gaseous swelling should be saved in a material property.
`save_solid_swell`	A flag indicating whether the solid swelling should be saved in a material property.

16.2.28 ZrPhase

The `ZrPhase` model computes the volume fraction of β phase for Zr-based cladding materials as a function of temperature and time.

```
[./zrphase]
  type = ZrPhase
  block = <string list>
  temperature = <string>
  numerical_method = <integer>
[../]
```

`type`	ZrPhase
`block`	List of blocks this material model applies to.
`temperature`	**Required**. Name of temperature variable. Typically `temp`.
`numerical_method`	Choice of numerical method to solve the differential equation for the phase transformation rate. One of 1 (implicit Euler method) or 2 (second order Adams-Moulton method - AM2).

16.3 Mass Diffusion Models

This material computes a two-term Arrhenius diffusion coefficient of the form

$$d = d_1 \exp\left(\frac{-q_1}{RT}\right) + d_2 \exp\left(\frac{-q_2}{RT}\right). \tag{16.2}$$

```
[./arrheniusdiffusioncoef]
  type = ArrheniusDiffusionCoef
  d1 = <real> (5.6e-8)
  d1_function = <string>
  d1_function_variable = <string>
  d2 = <real> (5.2e-4)
```

111

```
   q1 = <real> (2.09e5)
   q2 = <real> (3.62e5)
   gas_constant = <real> (8.3143)
   temp = <string>
 [../]
```

type	ArrheniusDiffusionCoef
d1	First coefficient (m^2/2).
d1_function	Function to be multiplied by d1.
d1_function_variable	Variable to be used when evaluating d1_function. If not given, time will be used.
d2	Second coefficient (m^2/2).
q1	First activation energy (J/mol).
q2	Second activation energy (J/mol).
gas_constant	Universal gas constant (J/mol/K).
temp	Name of temperature variable. Typically temp.

16.4 Other Models

16.4.1 Arrhenius Material Property

ArrheniusMaterialProperty is used to declare an arbitrary material property D that has the form $D = Ae^{-Q/RT}$, where A is the frequency factor, Q is the activation energy, R is the gas constant, and T is the temperature.

```
 [./some_property]
   type = ArrheniusMaterialProperty
   frequency_factor = <real>
   activation_energy = <real>
   gas_constant = <real> (8.314)
   temp = <variable>
   property_name = <string>
 [../]
```

type	ArrheniusMaterialProperty
frequency_factor	The coefficient in front of the exponential.
activation_energy	The activation energy.
gas_constant	The universal gas constant.
temp	Coupled temperature variable.
property_name	The name for this property.

16.4.2 Density

The `Density` model creates a material property named density. If coupled to displacement variables, the model adjusts density based on deformation.

```
[./density]
  type = Density
  disp_x = <string>
  disp_y = <string>
  disp_z = <string>
  disp_r = <string>
  density = <real>
[../]
```

`type`	Density
`disp_x`	Variable name for displacement variable in x direction. Typically `disp_x`.
`disp_y`	Variable name for displacement variable in y direction. Typically `disp_y`.
`disp_z`	Variable name for displacement variable in z direction. Typically `disp_z` for 3D and `disp_y` for axisymmetric models.
`disp_r`	Variable name for displacement variable in radial direction for axisymmetric or spherically symmetric cases. Typically `disp_x`.
`density`	**Required**. Density.

17 Fission Gas Models

Fission gas production and release modeling plays a vital role in fuel performance analysis. Fission gas affects swelling, porosity, thermal conductivity, gap conductivity, and rod internal pressure. The `Sifgrs` model is recommended.

17.1 Sifgrs

`Sifgrs` is the recommended fission gas model. It computes both fission gas release and fuel gaseous swelling.

```
[./sifgrs]
  type = Sifgrs
  initial_porosity = <real> (0.05)
  grain_radius_const = <real> (5.e-06)
  hydrostatic_stress_const = <real> (0.)
  saturation_coverage = <real> (0.5)
  ath_model = <bool> (false)
  gbs_model = <bool> (false)
  hbs_model = <bool> (false)
  grain_radius = <string>
  temp = <string>
  fission_rate = <string>
  burnup = <string>
  burnup_function = <string>
  hydrostatic_stress = <string>
  pellet_id = <string>
  pellet_brittle_zone = <string>
  eff_diff_coeff_option = <integer> (0)
  diff_coeff_option = <integer> (3)
  res_param_option = <integer> (0)
  trap_param_option = <integer> (0)
  ig_fully_coupled = <integer> (0)
  ig_bubble_model = <integer> (0)
  ig_diff_algorithm = <integer> (0)
  transient_option = <integer> (0)
  rod_ave_lin_power = <string>
  axial_power_profile = <string>
  temperature_scalef = <real> (1.)
  grainradius_scalef = <real> (1.)
  effdiffcoeff_scalef = <real> (1.)
  igdiffcoeff_scalef = <real> (1.)
  resolutionp_scalef = <real> (1.)
```

```
   trappingp_scalef = <real> (1.)
   gbdiffcoeff_scalef = <real> (1.)
 [../]
```

type	Sifgrs
initial_porosity	Initial fuel porosity (/).
grain_radius_const	A constant value for fuel grain radius (m). Ignored if grain_radius is given.
hydrostatic_stress_const	A constant value for hydrostatic stress (Pa). Ignored if hydrostatic_stress is given.
saturation_coverage	Initial value of fractional grain boundary bubble coverage at saturation (/).
ath_model	Whether to consider athermal (recoil and knockout) gas release.
gbs_model	Whether to consider the grain boundary sweeping effect.
hbs_model	Whether to consider intra-granular gas depletion in the high burnup structure (HBS).
grain_radius	Name of the auxiliary variable for grain radius, calculated by GrainRadiusAux (Section 12.3.5). Typically grain_radius.
temp	Variable name for temperature variable. Typically temp.
fission_rate	Variable name corresponding to the fission rate. Typically fission_rate.
burnup	Name of burnup variable. Typically burnup.
burnup_function	Name of the Burnup sub-block, if any (typically burnup). May not be used with fission_rate. Use of burnup_function is preferred.
hydrostatic_stress	Variable name for hydrostatic stress. Typically hydrostatic_stress.
pellet_id	Name of the auxiliary variable for pellet id. Typically pellet_id. Used if ath_model = true.
pellet_brittle_zone	The name of the UserObject that computes the width of the brittle zone in the fuel pellet. Used if ath_model = true.
eff_diff_coeff_option	Choice of the intra-granular effective diffusion coefficient. One of 0 (based on Speight formulation [11]), or 1 (correlation from [12]).
diff_coeff_option	Choice of the intra-granular diffusion coefficient. One of 0 (Turnbull et al. [13] with purely irradiation-dependent term not considered), 1 (Andersson et al. [14]), 2 (Turnbull et al. [15]), or 3 (Turnbull et al. [15] with purely irradiation-dependent term not considered).

res_param_option	Choice of the intra-granular resolution parameter. One of 0 (heterogeneous model [16]), 1 (homogeneous model from [17]), or 2 (reference value for sensitivity analysis as in [18]).
trap_param_option	Choice of the intra-granular trapping parameter. Only the option 0 (based on [19]) is currently available.
ig_fully_coupled	Not currently available.
ig_bubble_model	Choice of the intra-granular bubble model. One of 0 (empirical model from [16]), or 1 (constant values from [20]).
ig_diff_algorithm	Choice of the algorithm to solve the intra-granular diffusion equation. One of 0 (algorithm from [21]), or 1 (more recent and accurate algorithm PolyPole-1 [22]).
transient_option	To consider the burst release effect during transients, set equal to 2.
rod_ave_lin_pow	Function describing rod averaged linear power.
axial_power_profile	Function describing axial power profile.
temperature_scalef	Scale factor applied to temperature (for sensitivity studies).
grainradius_scalef	Scale factor applied to grain radius (for sensitivity studies).
effdiffcoeff_scalef	Scale factor applied to intra-granular effective diffusion coefficient (for sensitivity studies).
igdiffcoeff_scalef	Scale factor applied to intra-granular atomic diffusion coefficient (for sensitivity studies).
resolutionp_scalef	Scale factor applied to intra-granular resolution parameter (for sensitivity studies).
trappingp_scalef	Scale factor applied to intra-granular trapping parameter (for sensitivity studies).
gbdiffcoeff_scalef	Scale factor applied to grain-boundary diffusion coefficient (for sensitivity studies).

17.2 ForMas

The ForMas model is maintained but not actively developed. The Sifgrs model is recommended.

```
[./formas]
  type = ForMas
  grain_radius = <real> (10e-6)
  resolution_rate = <real> (1e-7)
  resolution_depth = <real> (1e-8)
  bubble_radius = <real> (5e-7)
  bubble_shape_factor = <real> (0.287)
```

116

```
    surface_tension = <real> (0.626)
    fractional_coverage = <real> (0.5)
    external_pressure = <real> (10e6)
    plenum_pressure = <string>
    external_pressure_function = <string>
    release_fraction = <real> (0)
    fractional_yield = <real> (0.3017)
    calibration_factor = <real> (1)
  [../]
```

type	ForMas
grain_radius	Initial fuel grain radius.
resolution_rate	Resolution rate from intergranular bubbles (1/s).
resolution_depth	Resolution layer depth.
bubble_radius	Grain boundary bubble radius.
bubble_shape_factor	Non-spherical bubble shape factor.
surface_tension	Bubble surface tension (J/m^2).
fractional_coverage	Fractional coverage of grain boundary at saturation.
external_pressure	Constant external hydrostatic pressure.
plenum_pressure	The name of the plenum pressure Postprocessor.
external_pressure_function	Function describing the external pressure.
release_fraction	Fraction of boundary and resolved gas released at saturation.
fractional_yield	Fractional yield of fission gas atoms per fission.
calibration_factor	Calibration factor to be multiplied by gas saturation density.

17.3 FgrUPuZr

FgrUPuZr is a fission gas release model for UPuZr metal fuel.

```
  [./upuzr_fission_gas_release]
    type = FgrUPuZr
    fission_rate = <string>
    fractional_yield = <real> (0.3017)
  [../]
```

type	FgrUPuZr
fission_rate	Coupled Fission Rate
fractional_yield	Fractional yield of fission gas atoms per fission.

18 Postprocessors

MOOSE `Postprocessors` compute a single scalar value at each timestep. These can be minimums, maximums, averages, volumes, or any other scalar quantity. One example of the use of `Postprocessors` in BISON is computing the gas volume of an LWR rod. The gas volume changes timestep to timestep, but since it is a single scalar quantity, a `Postprocessor` computes this value.

```
[Postprocessors]
  [./name]
    type = <postprocessor type>
    block = <string list>
    boundary = <string list>
    execute_on = <string list>
    outputs = <string>
    ...
  [../]
[]
```

`type`	Type of postprocessor
`block`	List of blocks. Either block numbers or names.
`boundary`	List of boundaries (side sets). Either boundary numbers or names.
`execute_on`	Set to (`nonlinear`—`linear`—`timestep_end`—`timestep_begin`) to execute at that moment.
`outputs`	Vector of output names where you would like to restrict the output of variable(s) associated with the postprocessor.

Most `Postprocessors` act on either boundaries or blocks. If no block or boundary is specified, the `Postprocessor` will act on the entire model. There are a few `Postprocessors` that act on specific nodes or elements within the finite element mesh.

18.1 DecayHeatFunction

`DecayHeatFunction` computes the value of the decay heat function. The value is zero prior to the specified `time_at_shutdown`. This postprocessor is typically used for Loss of Coolant Accident simulations.

```
  [./decayheatfunction]
    type = DecayHeatFunction
    energy_per_fission = <real> (3.28451e-11)
```

```
   neutron_capture_factor = <real> (1)
   time_at_shutdown = <real> (1e10)
 [../]
```

type	DecayHeatFunction
energy_per_fission	The energy released per fission in J/fission.
neutron_capture_factor	The neutron capture factor used to account for the effect of neutron capture in fission products.
time_at_shutdown	The time the reactor is shutdown and decay heat begins to take effect.

18.2 ElementIntegralPower

ElementIntegralPower computes the power in the supplied block given the fission rate variable and energy per fission.

```
 [./elementintegralpower]
   type = ElementIntegralPower
   burnup_function = <string>
   fission_rate = <string>
   energy_per_fission = <real> (3.28451e-11)
   variable = <string>
 [../]
```

type	ElementIntegralPower
burnup_function	Name of the Burnup sub-block, if any (typically burnup). May not be used with fission_rate. Use of burnup_function is preferred.
fission_rate	Variable name corresponding to the fission rate. Typically fission_rate.
energy_per_fission	The energy released per fission in J/fission.
variable	The variable name this Postprocessor applies to. Typically temp.

18.3 ElementalVariableValue

In some cases it may be of interest to output an elemental variable value (e.g., stress) at a particular location in the model. This is accomplished by using the ElementalVariableValue postprocessor.

```
 [./elementalvariablevalue]
   type = ElementalVariableValue
   elementid = <string>
   variable = <string>
 [../]
```

`type`	`ElementalVariableValue`
`elementid`	**Required**. The global element id from the mesh to which this postprocessor is to be applied.
`variable`	**Required**. The variable whose value is output to this postprocessor for the given element.

18.4 Fission Gas Postprocessors

When using the Sifgrs fission gas release model there are four postprocessors that are used to report the fission gas that is produced in moles (`ElementIntegralFisGasGeneratedSifgrs`), fission gas within the grains (`ElementIntegralFisGasGrainSifgrs`), fission gas on the grain boundary (`ElementIntegralFisGasBoundarySifgrs`), and the fission gas released to the plenum in moles (`ElementIntegralFisGasReleasedSifgrs`). The details of including these postprocessors in the input file is outlined below:

```
[./fis_gas_produced]
  type = ElementIntegralFisGasGeneratedSifgrs
  variable = <string>
  block = <string list>
[../]

[./fis_gas_grain]
  type = ElementIntegralFisGasGrainSifgrs
  variable = <string>
  block = <string list>
[../]

[./fis_gas_boundary]
  type = ElementIntegralFisGasBoundarySifgrs
  variable = <string>
  block = <string list>
[../]

[./fis_gas_released]
  type = ElementIntegralFisGasReleasedSifgrs
  variable = <string>
  block = <string list>
[../]
```

`type`	The type of postprocessor
`variable`	**Required**. The variable the postprocessor applies to. For these fission gas postprocessors the variable is typically `temp`.
`block`	The blocks the postprocessor applies to. For nuclear fuel simulations fission gas calculations apply to the fuel/pellet block.

18.5 IFBAHeProduction

IFBAHeProduction computes the Helium gas production as a result of an IFBA layer applied to the surface of a fuel rod.

```
[./he_prod]
  type = IFBAHeProduction
  zrb2_load = <real>
  b10_load = <real>
  ifba_len = <real>
  b10_enrich = <real>
  zrb2_rel_dens = <real>
  zrb2_thk = <real>
  fuel_out_rad = <real>
  model = <string>
  u235_enrich = <real>
  ifba_rod_pct = <real>
  burnup = <string>
  rod_ave_lin_pow = <string>
[../]
```

type	IFBAHeProduction
zrb2_load	ZrB$_2$ loading of IFBA layer (kg/m). (Cannot be used with b10_load)
b10_load	B-10 loading of IFBA layer (kg/m). (Cannot be used with zrb2_load)
ifba_len	**Required**. The length of the IFBA layer (m).
b10_enrich	**Required**. B-10 enrichment in atomic percent (fraction).
zrb2_rel_dens	ZrB$_2$ relative density (fraction). (Cannot be used with either zrb2_thk or fuel_out_rad)
zrb2_thk	ZrB$_2$ layer thickness (m). Must also specify fuel_out_rad. (Cannot be used with zrb2_dens)
fuel_out_rad	Outer radius of fuel (m). Must also specify zrb2_thk. (Cannot be used with zrb2_dens)
model	Specifies which equation to use for Helium generation (string). burnup = Burnup based equation, frapcon = FRAPCON equation.
u235_enrich	U-235 enrichment of the fuel in atomic percent (fraction). (Only used with model = burnup)
burnup	Postprocessor providing the average burnup. (Only used with model = burnup, typically average_burnup)
ifba_rod_pct	Percentage of IFBA rods in a fuel assembly (fraction). (Only used with model = frapcon)

rod_ave_lin_pow   Postprocessor providing the rod linear power.
                   (Only used with `model = frapcon`, typically `power_history`)

18.6 InternalVolume

`InternalVolume` computes the volume of an enclosed space. The entire boundary of the enclosed space must be represented by the given side set. If the given side set points outward, `InternalVolume` will report a negative volume.

```
[./internalvolume}
  type = InternalVolume
  scale_factor = <real> (1)
  addition = <addition> (0)
[../]
```

type	InternalVolume
scale_factor	Scale factor to be applied to the internal volume calculation.
addition	Number to be added to internal volume calculation. This addition is not scaled.

18.7 MaterialTimeStep

`MaterialTimeStep` allows one to apply physics-based time step criteria implemented in material models. For this purpose, this postprocessor takes the overall minimum across the domain of the time steps provided by (one or more) material models. At this time, a creep rate based time step criterion is available in `MechZry` (Section 16.2.18). `MaterialTimeStep` will retrieve the time step length, which `MechZry` computes at each mesh point, and return the minimum. This time step can then be used by a time stepper such as `PostprocessorDT`. `MaterialTimeStep` can be extended to consider multiple material model time step criteria as they become available.

```
[./materialtimestep}
  type = MaterialTimeStep
  variable = <string>
[../]
```

type	MaterialTimeStep
variable	**Required**. The variable whose value is output to this postprocessor.

18.8 NodalVariableValue

In order to obtain the value of a nodal variable at a particular location (i.e., temperature and displacement) a `NodalVariableValue` postprocessor is used. For example, this postprocessor

122

is useful for obtaining the centerline temperature at the location of a thermocouple to compare against experimental data.

```
[./nodalvariablevalue]
  type = NodalVariableValue
  elementid = <string>
  scale_factor = <real>
  variable = <string>
[../]
```

type	NodalVariableValue
nodeid	**Required**. The global node id from the mesh to which this postprocessor is to be applied.
scale_factor	A scalar value to be multiplied by the value of the variable.
variable	**Required**. The variable whose value is output to this postprocessor for the given node.

18.9 NumNonlinearIterations

NumNonlinearIterations reports the number of nonlinear iterations in the just-completed solve.

```
[./numnonlineariters]
  type = NumNonlinearIterations
[../]
```

type	NumNonlinearIterations

18.10 PlenumTemperature

PlenumTemperature gives an estimate of the temperature of the gas in the plenum. This is done by computing a volume weighted average temperature using gap sizes between surfaces. A typical use is to pair a set of fuel surfaces (inner_surfaces) with a corresponding set of cladding surfaces (outer_surfaces). Another possible use is to pair the inner radius of the cladding (inner_surfaces) with the outer radius of the fuel (outer_surfaces). It is also possible to pair up surfaces automatically for a discrete pellet mesh using the num_pellets option.

```
[PlenumTemperature]
  [./plenumtemperature]
    boundary = <string list>
    inner_surfaces = <string list>
    outer_surfaces = <string list>
    inner_radius_surfaces = <string list>
    temp = <string>
```

```
      num_pellets = <integer>
      order = <string> (FIRST)
    [../]
  []
```

`boundary`	**Required**. List of boundaries (side sets). Either boundary numbers or names. The calculation of the temperature will occur along these boundaries.
`inner_surfaces`	List of boundaries. Will be paired with `outer_surfaces` to determine the temperature on the opposite surface.
`outer_surfaces`	List of boundaries. Will be paired with `inner_surfaces`.
`inner_radius_surfaces`	List of boundaries. These are typically fuel surfaces along the inner radius of annular fuel pellets.
`temp`	**Required**. Variable name for temperature. Typically `temp`.
`num_pellets`	Number of discrete pellets in the model. Cannot be used with `inner_surfaces`, `outer_surfaces`, and `inner_radius_surfaces`.
`order`	The order of the variables used. Typical values are `FIRST` and `SECOND`.

18.11 PlotFunction

`PlotFunction` gives the value of the supplied function at the current time, optionally scaled with `scale_factor`.

```
  [./plotfunction]
    type = PlotFunction
    function = <string>
    scale_factor = <real> (1)
  [../]
```

`type`	`PlotFunction`
`function`	**Required**. The function to evaluate.
`scale_factor`	Scale factor to be applied to the function value.

18.12 SideAverageValue

`SideAverageValue` computes the area- or volume-weighted average of the named variable. It may be used, for example, to calculate the average temperature over a side set.

```
  [./sideaveragevalue}
    type = SideAverageValue
```

```
    variable = <string>
  [../]
```

type   SideAverageValue

variable   **Required**. The variable this Postprocessor acts on.

18.13 SideFluxIntegral

SideFluxIntegral computes the integral of the flux over the given boundary.

```
  [./sidefluxintegral]
    type = SideFluxIntegral
    variable = <string>
    diffusivity = <string>
  [../]
```

type   SideFluxIntegral

variable   **Required**. Variable to be used in the flux calculation.

diffusivity   **Required**. The diffusivity material property to be used in the calculation.

18.14 TimestepSize

TimestepSize reports the timestep size.

```
  [./dt]
    type = TimestepSize
  [../]
```

type   TimestepSize

19 Solution Execution and Time Stepping

The `Executioner` block describes how the simulation will be executed. It includes commands to control the solver behavior and time stepping. Time stepping is controlled by a combination of commands in the `Executioner` block, and the `TimeStepper` block nested within the `Executioner` block.

```
[Executioner]
  type = <string>
  solve_type = <string>
  petsc_options = <string list>
  petsc_options_iname = <string list>
  petsc_options_value = <string list>
  line_search = <string>
  l_max_its = <integer>
  l_tol = <real>
  nl_max_its = <integer>
  nl_rel_tol = <real>
  nl_abs_tol = <real>
  start_time = <real>
  dt = <real>
  end_time = <real>
  num_steps = <integer>
  dtmax = <real>
  dtmin = <real>
  [TimeStepper]
    #TimeStepper commands
  [../]
```

type	**Required**. Several available. Typically `Transient`.
solve_type	One of `PJFNK` (preconditioned JFNK), `JFNK` (JFNK), `NEWTON` (Newton), or `SolveFD` (Jacobian computed by finite difference– serial only, slow).
petsc_options	PETSc flags.
petsc_options_iname	Names of PETSc name/value pairs.
petsc_options_value	Values of PETSc name/value pairs.
line_search	Line search type. Typically `none`.
l_max_its	Maximum number of linear iterations per solve.
l_tol	Linear solve tolerance.
nl_max_its	Maximum number of nonlinear iterations per solve.

nl_rel_tol	Nonlinear relative tolerance.
nl_rel_abs	Nonlinear absolute tolerance.
start_time	The start time of the analysis.
end_time	The end time of the analysis.
num_steps	The maximum number of time steps.
dtmax	The maximum allowed timestep size.
dtmin	The minimum allowed timestep size.

Several `Executioner` types exist, although the `Transient` type is typically the appropriate one to use for transient BISON analyses. For each type, specific options are available. To see the complete set of possibilities, follow the Input Syntax link on the BISON wiki page.

Similarly, many PETSc options exist. Please see the online PETSc documentation for details.

Given the many possibilities in the `Executioner` block, it may be helpful to review examples in the BISON tests, examples, and assessment directories.

19.1 Timestepping

The method used to calculate the size of the time steps taken by BISON is controlled by the `TimeStepper` block. There are a number of types of `TimeStepper` available. Three of the types most commonly used with BISON are described here. These permit the time step to be controlled directly by providing either a single fixed time step to take throughout the analysis, by providing the time step as a function of time, or by using adaptive timestepping algorithm can be used to modify the time step based on the difficulty of the iterative solution, as quantified by the numbers of linear and nonlinear iterations required to drive the residual below the tolerance required for convergence.

19.1.1 Direct Time Step Control with Constant Time Step

The `ConstantDT` type of `TimeStepper` simply takes a constant time step size throughout the analysis.

```
[TimeStepper]
  type = ConstantDT
  dt = <real>
[../]
```

type	ConstantDT
dt	**Required**. The initial timestep size.

`ConstantDT` begins the analysis taking the step specified by the user with the `dt` parameter. If the solver fails to obtain a converged solution for a given step, the executioner cuts back the step size and attempts to advance the time from the previous step using a smaller time step. The time step is cut back by multiplying the time step by 0.5.

If the solution with the cut-back time step is still un-successful, it is repeatedly cut back until a successful solution is obtained. The user can specify a minimum time step through the `dtmin` parameter in the `Executioner` block. If the time step must be cut back below the minimum size without obtaining a solution, BISON exits with an error. If the time step is cut back using `ConstantDT`, that cut-back step size will be used for the remainder of the the analysis.

19.1.2 Direct Time Step Control with Varying Time Step Size

If the `FunctionDT` type of `TimeStepper` is used, BISON takes time steps that vary over time according to a user-defined function.

```
[TimeStepper]
  type = FunctionDT
  time_t = <real list>
  time_dt = <real list>
[../]
```

type	FunctionDT
time_t	The abscissas of a piecewise linear function for timestep size.
time_dt	The ordinates of a piecewise linear function for timestep size.

The time step is controlled by a piecewise linear function defined using the $time\_t$ and $time\_dt$ parameters. A vector of time steps is provided using the $time\_dt$ parameter. An accompanying vector of corresponding times is specified using the $time\_t$ parameter. These two vectors are used to form a time step vs. time function. The time step for a given step is computed by linearly interpolating between the pairs of values provided in the vectors.

The same procedure that is used with `ConstantDT` is used to cut back the time step from the user-specified value if a failed solution occurs.

19.1.3 Adaptive Time Stepping

The `IterationAdaptiveDT` type of `TimeStepper` provides a means to adapt the time step size based on the difficulty of the solution.

```
[TimeStepper]
  type = IterationAdaptiveDT
  dt = <real>
  optimal_iterations = <integer>
  iteration_window = <integer> (0.2*optimal_iterations)
  linear_iteration_ratio = <integer> (25)
  growth_factor = <real>
  cutback_factor = <real>
  timestep_limiting_function = <string>
  max_function_change = <real>
  force_step_every_function_point = <bool> (false)
[../]
```

dt	**Required**. The initial timestep size.
optimal_iterations	The target number of nonlinear iterations for adaptive timestepping.
iteration_window	The size of the nonlinear iteration window for adaptive timestepping.
linear_iteration_ratio	The ratio of linear to nonlinear iterations to determine target linear iterations and window for adaptive timestepping.
growth_factor	Factor by which timestep is grown if needed.
cutback_factor	Factor by which timestep is cut back if needed.
timestep_limiting_function	Function used to control the timestep.
max_function_change	Maximum change in the function over a time step.
force_step_every_function_point	Controls whether a step is forced at every point in the function.

IterationAdaptiveDT grows or shrinks the time step based on the number of iterations taken to obtain a converged solution in the last converged step. The required optimal_iterations parameter controls the number of nonlinear iterations per time step that provides optimal solution efficiency. If more iterations than that are required to obatin a converged solution, the time step may be too large, resulting in undue solution difficulty, while if fewer iterations are required, it may be possible to take larger time steps to obtain a solution more quickly.

A second parameter, iteration_window, is used to control the size of the region in which the time step is held constant. As shown in Figure 19.1, if the number of nonlinear iterations for convergence is lower than (optimal_iterations−iteration_window), the time step is increased, while if more than (optimal_iterations+iteration_window), iterations are required, the time step is decreased. The iteration_window parameter is optional. If it is not specified, it defaults to 1/5 the value specified for optimal_iterations.

The decision on whether to grow or shrink the time step is based both on the number of nonlinear iterations and the number of linear iterations. The parameters mentioned above are used to control the optimal iterations and window for nonlinear iterations. The same criterion is applied to the linear iterations. Another parameter, linear_iteration_ratio, which defaults to 25, is used to control the optimal iterations and window for the linear iterations. These are calculated by multiplying linear_iteration_ratio by optimal_iterations and iteration_window, respectively.

To grow the time step, the growth criterion must be met for both the linear iterations and nonlinear iterations. If the time step shrinkage criterion is reached for either the linear or nonlinear iterations, the time step is decreased. To control the time step size only based on the number of nonlinear iterations, set linear_iteration_ratio to a large number.

If the time step is to be increased or decreased, that is done using the factors specified with the growth_factor and cutback_factor, respectively. If a solution fails to converge when adaptive time stepping is active, a new attempt is made using a smaller time step in the same manner as with the fixed time step methods. The maximum and minimum time steps can be optionally

specified in the `Executioner` block using the `dtmax` and `dtmin` parameters, respectively.

In addition to controlling the time step based on the iteration count, `IterationAdaptiveDT` also has an option to limit the time step based on the behavior of a time-dependent function, optionall specified by providing the function name in `timestep_limiting_function`. This is typically a function that is used to drive boundary conditions of the model. The step is cut back if the change in the function from the previous step exceeds the value specified in `max_function_change`. This allows the step size to be changed to limit the change in the boundary conditions applied to the model over a step. In addition to that limit, the boolean parameter `force_step_every_function_point` can be set to `true` to force a time step at every point in a `PiecewiseLinear` function.

Figure 19.1: Criteria used to determine adaptive time step size

19.2 PETSc Options

The amount of PETSc options to choose as solver parameters is vast and cannot be covered in detailed here. This section provides the recommended PETSc options depending upon whether Dirac or Constraint based contact is used. The values for the `petsc_options_value` can change depending on the particular problem being analyzed. For specialized problems where these standard options do not work the user is encouraged to consult the PETSc User's Manual or contact the `bison-users` mailing list.

19.2.1 Constraint Contact

The recommended PETSc options for use with Constraint based contact are given below:

```
[Executioner]
  ...
  petsc_options_iname = '-pc_type -sub_pc_type -pc_asm_overlap
                          -ksp_gmres_restart'
  petsc_options_value = 'asm lu 20 101'
  ...
[../]
```

19.2.2 Dirac Contact

The recommended PETSc options for use with Dirac based contact are given below:

```
[Executioner]
  ...
  petsc_options_iname = '-ksp_gmres_restart -pc_type -pc_hypre_type
                         -pc_hypre_boomeramg_max_iter'
  petsc_options_value = '201 hypre boomeramg 4'
  ...
[../]
```

19.3 Quadrature

When using higher order meshes (e.g. second) it is recommended to use `quadrature = true` in the thermal contact block. When this parameter is set the order of the quadrature can be specified using a `[./Quadrature]` subblock within the `[Executioner]` block as follows:

```
[./Quadrature]
  type = <string>
  element_order = <string>
  order = <string>
  side_order = <string>
[../]
```

`type`	The type of quadrature used. Default is Gauss.
`element_order`	Order of quadrature on the elements.
`order`	Order of quadrature used.
`side_order`	Order of quadrature used on the sides.

The recommended `[./Quadrature]` block when using second-order meshes is the following:

```
[./Quadrature]
  order = FIFTH
  side_order = SEVENTH
[../]
```

131

20 Outputs

The `Outputs` block lists parameters that control the frequency and type of results files produced. It is possible to create multiple output objects each outputting at different interals, or different variables, or varying file types. The `Outputs` system is vary complex and enables a large amount of customization. This section will highlight different capabilities of the system. At the end of the this section an example of a typical `Outputs` block for BISON assessment cases will be presented.

20.1 Basic Input File Syntax

To enable output an input file must contain an `Outputs` block. The simplest method for enabling output is to utilize the shortcut syntax as shown below, which enables the `Console` output (prints to screen) and `Exodus` output for writing data to a file.

```
[Outputs]
  console = true    #output to the screen with default settings
  exodus = true     #output to ExodusII file with default settings
[]
```

20.2 Advanced Syntax

To take full advantage of the output system the use of subblocks is required. For example, the input file snippet below is **exactly** equivalent, including the subblock names, to the snippet shown above that utitlizes the shortcut syntax.

```
[Outputs]
  [./console]
    type = Console    #output to the screen with default settings
  [../]
  [./exodus]
    type = Exodus     #output to ExodusII file with default settings
  [../]
[]
```

However, the subblock syntax allows for increased control over the output and allows for multiple outputs of the same type to be specified. For example, the following creates two `Exodus` outputs, one outputting the a mesh at every time step including the initial condition the other outputs every 3 time steps without the initial condition. Additionally, performance logging was enabled for `Console` output.

```
[Outputs]
  [./console]
    type = Console
    perf_log = true        # enable performance logging
  [../]
  [./exodus]
    type = Exodus
    output_initial = true  # enable the output of the initial
                           # condition for the [ExodusII][1] file
  [../]
  [./exodus_3]                 # create a second [Exodus II][1] output
                               # that utilizes a different output interval
    type = Exodus
    file_base = exodus_3   # set the file base
                           # (the extension is automatically applied)
    interval = 3           # only output every third step
  [../]
[]
```

20.3 Common Output Parameters

In addition to allowing for short-cut syntax, the Outputs block also supports common parameters. For example, output_initial may be specified outside of individual subblocks, indicating that all subblocks should output the initial condition. If within a subblock the parameter is given a different value, the subblock parameter takes precedence. The input file snippet below demonstrate the usage of a common values as well as the use of multiple output blocks.

```
[Outputs]
  output_initial = true      # set all subblocks to output the
                             # initial condition
  vtk = true                 # output VTK file with default setting
  [./console]
    type = Console
    perf_log = true
  [../]
  [./exodus]
    type = Exodus
    output_initial = false   # this ExodusII files will not contain
                             # the initial condition
  [../]
[]
```

20.4 File Output Names

The default naming scheme for output files utilizes the input file name (e.g., input.i) with a suffix that differs depending on how the output is defined:

- outputs create using the shortcut syntax an "_out" suffix is utilized and

- subblocks use the actual subblock name as the suffix.

For example, if the input file (input.i) contained the following `[Outputs]` block, two files would be created: input_out.e and input_other.e.

```
[Outputs]
  console = true
  exodus = true     # creates input_out.e
  [./other]         # creates input_other.e
    type = Exodus
    interval = 2
  [../]
[]
```

20.5 Typical BISON Example

Now that some of the basic capabilities of the output system have been outlined, a typical `[Outputs]` block from a BISON assessment case is presented.

```
[Outputs]
  interval = 1
  output_initial = true
  csv = true
  exodus = true
  color = false
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
    max_rows = 25
  [../]
[]
```

`interval`	The interval at which timesteps are output to the solution file. This is a global output parameter since it is not in a subblock.
`output_initial`	Request that the initial condition is output to the solution file. This is a global output parameter since it is not in a subblock.
`csv`	Specify that a csv file be output containing values of all postprocessors.
`exodus`	Specify that an ExodusII file be output.

`color`	Specify that color not be output to the screen for the log.
`type`	Specify the type for the subblock. In this case Console.
`perf_log`	Specify that the performance log be output to the screen.
`linear_residuals`	Specify that the linear residuals be output to the screen.
`max_rows`	The maximum number of postprocessor/scalar values displayed on the screen during a timestep (set to 0 for unlimited).

21 Dampers

Dampers are used to decrease the attempted change to the solution with each nonlinear step. This can be useful in preventing the solver from changing the solution dramatically from one step to the next. This may prevent, for example, the solver from attempting to evaluate negative temperatures.

The `MaxIncrement` damper is commonly used.

21.1 MaxIncrement

The `MaxIncrement` damper limits the change of a variable from one nonlinear step to the next.

```
[Dampers]
  [./maxincrement]
    type = MaxIncrement
    max_increment = <real>
    variable = <string>
  [../]
[]
```

`type`	MaxIncrement
`max_increment`	**Required**. The maximum change in solution variable allowed from one nonlinear step to the next.
`variable`	**Required**. Variable that will not be allowed to change beyond `max_increment` from nonlinear step to nonlinear step.

22 Restart and Recover

The MOOSE framework provides two ways of continuing a simulation: recover and restart. An example restart problem is located at `projects/bison/examples/restart`. The instructions below are copied from the MOOSE Wiki.

22.1 Definitions

- **Restart:** Running a simulation that uses data from a previous simulation. Data in this context is very broad, it can mean spatial field data, non-spatial variables or postprocessors, or stateful object data. Usually the previous and new simulations use different input files.

- **Recover:** Resuming an existing simulation either due to a fault or other premature termination.

- **Solution File:** A mesh format containing field data in addition to the mesh (i.e. a normal output file).

- **Checkpoint:** A snapshot of the simulation data including all meshes, solutions, and stateful object data. Typically one checkpoint is stored in several different files.

- **N to N:** In a restart context, this means the number of processors for the previous and current simulations must match.

- **N to M:** In a restart context, different numbers of processors may be used for the previous and current simulations.

22.2 Simple Restart (Variable initialization)

- This method is best suited for restarting a simulation when the mesh in the previous simulation exactly matches the mesh in the current simulation and only initial conditions need to be set for one more variables.

- This method requires only a valid Solution File.

- MOOSE supports N to M restart when using this method.

```
# Reading field data from a nodal or elemental field from a
# previous simulation
[Mesh]
  # MOOSE supports reading field data from ExodusII, XDA/XDR, and
  # mesh checkpoint files (.e, .xda, .xdr, .cp)
  file = previous.e
  # This method of restart is only supported on serial meshes
  distribution = serial
[]

[Variables]
  [./nodal]
    family = LAGRANGE
    order = FIRST
    initial_from_file_var = nodal
    initial_from_file_timestep = 10
  [../]
[]

[AuxVariables]
  [./elemental]
    family = MONOMIAL
    order = CONSTANT
    initial_from_file_var = elemental
    initial_from_file_timestep = 10
  [../]
[]
```

22.3 Enabling Checkpoints

Advanced restart in MOOSE requires checkpoint files. To enable automatic checkpoints using the default options (every time step, and keep last two) in your simulation simply add the following flag to your input file:

```
[Outputs]
  checkpoint = true
[]
```

f you need more control over the checkpoint system, you can create a subblock in the input file that will allow you to change the file format, suffix, frequency of output, the number of checkpoint files to keep, etc. For a complete list see the Doxygen page for Checkpoint.

Note: You should always set num_files to at least 2 to minimize the change of ending up with a corrupt restart file.

```
[Outputs]
  [./my_checkpoint]
    type = Checkpoint
```

```
    num_files = 4
    interval = 5
  [../]
[]
```

22.4 Advanced Restart

- This method is best suited for situations when the mesh from the previous simulation and the current simulation match but all variables should be reloaded and all stateful data should be restored.

- Support for modifying some variables is supported such as dt and time_step. By default, MOOSE will automatically use the last values found in the checkpoint files.

- Only N to N restarts are supported using this method.

```
[Mesh]
  # Serial number should match corresponding Executioner parameter
  file = out_cp/0010_mesh.cpr
  # This method of restart is only supported on serial meshes
  distribution = serial
[]

[Executioner]
  type = Transient

  # Note that the suffix is left off in the parameter below.
  restart_file_base = out_cp/0010
[]
```

22.5 Reloading Data

It is possible to load and project data onto a different mesh from a solution file usually as an initial condition in a new simulation. MOOSE fully supports this through the use of SolutionUserObject (see Section 23.3).

22.6 Recover

Whenever MOOSE is being run with checkpoints enabled, a simulation that has terminated due to a fault can be recovered simply by using the −−recover CLI flag.

As a supplement to this example, also included is a restart.sh script (`bison/examples/restart`), which can serve as an example and reference for commands to use when using restart. The purpose of this script is to test the functionality of restart.

23 UserObjects

23.1 FuelPinGeometry

`FuelPinGeometry` computes the fuel rod outer dimensions, the height, ID, and OD of the fuel stack, and the pellet-clad gap directly from the mesh so that it can be used by other objects such as `Burnup`, `RelocationUO2`, and `HydrogenPickup`. Special attention to the sidesets needs to be made when constructing the mesh. Using the BISON standard sideset convention, FuelPin-Geometry uses sideset 8 (Default) to compute the dimensions of the rod. In most of the cases that can be found in the BISON repository sideset 8 only calculates the pellet volume, whereas sideset 9 is used to calculate the plenum gas volume and temperature. The BISON meshing script will assign these sidesets properly if a simple fuel stack configuration is used (ie. solid or annular fuel and cladding only). If insulator pellets are required the current meshing script will include them in sideset 8, thus the pellet volume will be high. The solution is to reassign the sidesets as needed. If a more complex mesh is required, it will have to be built by the user and the user would assign the sidesets. Checking the sidesets before running a simulation is a good practice.

```
  [./pin_geometry]
    type = FuelPinGeometry
    clad_inner_wall = <string>( "5" )
    clad_outer_wall = <string>( "2" )
    clad_top = <string>( "3" )
    clad_bottom = <string>( "1" )
    pellet_exteriors = <string>( "8" )
  [../]
```

type	FuelPinGeometry
clad_inner_wall	Sideset for inner wall of cladding, not including end caps.
clad_outer_wall	Sideset for outer wall of cladding.
clad_top	Sideset for top of cladding (top of upper end cap).
clad_bottom	Sideset for bottom of cladding (bottom of lower end cap).
pellet_exteriors	Sideset for all pellet exteriors.

23.2 PelletBrittleZone

`PelletBrittleZone` computes the brittle zone width on a per-pellet basis.

```
  [./pelletbrittlezone]
    type = PelletBrittleZone
    pellet_id = <string>
    temp = <string>
    pellet_radius = <real>
    a_lower = <real>
    a_upper = <real>
    number_pellets = <integer>
    fuel_pin_geometry = <string>
[../]
```

type	PelletBrittleZone
pellet_id	Variable name for pellet id. Typically pellet_id.
temp	Name of temperature variable. Typically temp.
pellet_radius	**Required if fuel_pin_geometry is not specified**. The outer radius of the fuel.
a_lower	**Required if fuel_pin_geometry is not specified**. The lower axial coordinate of the fuel stack.
a_upper	**Required if fuel_pin_geometry is not specified**. The upper axial coordinate of the fuel stack.
number_pellets	**Required**. Number of fuel pellets.
fuel_pin_geometry	Name of the FuelPinGeometry object (see 23.1).

23.3 SolutionUserObject

A solution user object reads a variable from a mesh in one simulation to another. In order to use a SolutionUserObject three additional parameters are required, an AuxVariable , a Function and an AuxKernel. The AuxVariable represents the variable to be read by the solution user object. The SolutionUserObject is set up to read the old output file. A SolutionFunction is required to interpolate in time and space the data from the SolutionUserObject. Finally, the FunctionAux is required that will query the function and write the value into the AuxVariable. An example of what additions are required to the input file is shown below:

```
[AuxVariables]
  [./temp]
  [../]
[]

[Functions]
  [./interpolated_temp]
    type = SolutionFunction
    from_variable = 'temp'
    solution = read_thermo_solution
  [../]
```

```
[]

[UserObjects]
  [./read_thermo_solution]
    type = SolutionUserObject
    mesh = 'temp_from_another_simulation.e'
    execute_on = 'residual'
    nodal_variables = 'temp'
  [../]
[]

[AuxKernels]
  [./interp_temp]
    type = FunctionAux
    variable = 'temp'
    function = 'interpolated_temp'
  [../]
[]
```

Note that in the SolutionUserObject subblock that the mesh parameter is **required**.

24 Reference Residual Problem

An advanced scenario that requires the addition of a `[Problem]` block in the input file is the `ReferenceResidualProblem`. Reference residual is an alternative way to signify convergence of a timestep. The structure of the `[Problem]` block for a two-dimensional axisymmetric simulation is as follows:

```
  [./referenceresidualproblem]
    coord_type = RZ
    type = ReferenceResidualProblem
    solution_variables = <string list>
    reference_residual_variables = <string list>
    acceptable_iterations = <integer> (0)
    acceptable_multiplier = <integer> (1)
[../]
```

type	ReferenceResidualProblem
solution_variables	Set of variables to be checked for relative convergences.
reference_residual_variables	Set of variables that provide reference residuals for the relative convergence check.
acceptable_iterations	Iterations after which convergence to acceptable limits are accepted.
acceptable_multiplier	Multiplier applied to relative tolerance for acceptable limit.

When using reference residual it is typically acceptable to loosen the relative tolerance for convergence by an order of magnitude. The difficulty in setting up a `ReferenceResidualProblem` currently is the requirement of creating an `AuxVariable` for each of the reference residual variables. Then for each `Kernel` that the corresponding solution variable applies to an additional line is required to save into the reference residual variable. This requires significant changes to the input file. If you would like to try using a `ReferenceResidualProblem`, please contact one of the BISON developers for more detailed instructions of setting it up.

The implementation of `ReferenceResidualProblem` is scheduled to be updated within the next year.

25 Frictional Contact Problem

Another advanced use of the [Problem] block is the FrictionalContactProblem. This is used when a user wants to use kinematic (default) enforcement of frictional contact. If a user wants to use the penalty method for frictional contact the friction_coefficient needs to be specified in the [Contact] block and the model parameter set to coulomb. A typical [Problem] block for a two-dimensional axisymmetric case is as follows:

```
[./frictionalcontactproblem]
  coord_type = RZ
  type = FrictionalContactProblem
  friction_coefficient = <real>
  master = <string list>
  slave = <string list>
  slip_factor = <real>
  slip_too_far_factor = <real>
  disp_x = <string>
  disp_y = <string>
  residual_x = <string>
  residual_y = <string>
  diag_stiff_x = <string>
  diag_stiff_y = <string>
  inc_slip_x = <string>
  inc_slip_y = <string>
  contact_slip_tolerance_factor = <real> (10)
  target_contact_residual = <real>
  maximum_slip_iterations = <integer> (100)
  minimum_slip_iterations = <integer> (1)
  slip_updates_per_iteration = <integer> (1)
  solution_variables = <string list>
  reference_residual_variables = <string list>
[../]
```

type	FrictionalContactProblem
friction_coefficient	**Required**. The friction coefficient applied between the interacting surfaces.
master	**Required**. Number or name IDs of the master surfaces for which slip should be calculated.
slave	**Required**. Number or name IDs of the slave surfaces for which slip should be calculated.

slip_factor	**Required**. The fraction of calculated slip to be applied for each interaction. A value of 1 means the entire amount of calculated slip is applied.
slip_too_far_factor	**Required**. The fraction of the calculated slip to be applied for each interaction that is in the slipped-too-far-state.
disp_x	**Required**. Variable name for the x-displacement. Typically disp_x.
disp_y	**Required**. Variable name for the y-displacement. Typically disp_y.
residual_x	**Required**. Name of auxiliary variable containing the saved x residual.
residual_y	**Required**. Name of auxiliary variable containing the saved y residual.
diag_stiff_x	**Required**. Name of auxiliary variable containing the saved x diagonal stiffness.
diag_stiff_y	**Required**. Name of auxiliary variable containing the saved y diagonal stiffness.
inc_slip_x	**Required**. Name of auxiliary variable used to store the incremental slip in the x direction.
inc_slip_y	**Required**. Name of auxiliary variable used to store the incremental slip in the y direction.
contact_slip_tolerance_factor	Multiplier on convergence criteria to determine when to start slipping.
target_contact_residual	Frictional contact residual convergence criterion.
target_relative_contact_residual	Frictional contact relative residual convergence criterion.
maximum_slip_iterations	Maximum number of slip iterations per step.
minimum_slip_iterations	Minimum number of slip iterations per step.
slip_updates_per_iteration	The number of slip updates per contact iteration.
solution_variables	Set of variables to be checked for relative convergences.
reference_residual_variables	Set of variables that provide reference residuals for the relative convergence check.

It can be seen that a signifcant amount of auxiliary variables are required to be added to the input file to make FrictionalContactProblem work. In addition references to saved variables as in the ReferenceResidualProblem case is also required. If you would like to use FrictionalContactProblem please contact a BISON developer for assistance. The implementation and robustness of FrictionalContactProblem is to be improved in the next year.

26 Mesh Script

26.1 Overview

To ease generation of LWR fuel meshes, a mesh script is available. The script relies on CU-BIT [6].

26.1.1 Run the Main Script

The mesh script is at `bison/tools/UO2/`. The main script (mesh_script.sh) is run from the shell command line. This script invokes the Python meshing script (mesh_script.py) and passes it an input file named mesh_script_input.py by default.

You invoke the script as:

```
> ./mesh_script.sh [-c -d -l] [-p path to mesh_script.py] [-i
   mesh_script_input.py] [-o output file name]
```

The `-c` flag will cause the script to check whether CUBIT can be loaded. The `-d` flag results in the deletion of the CUBIT journal file when the script completes. The `-l` flag will generate a log file (otherwise messages will go to the terminal). The `-p` flag, which is rarely used, tells the script where to find the mesh_script.py file. You may supply any mesh script input file with the `-i` flag. Finally, you may specify the name of the output Exodus file with the `-o` flag.

The main script generates an exodus file, with QUAD elements in 2D and HEX elements in 3D.

26.1.2 Mesh Architecture

Figure 26.1 provides an overview of the architecture of a fuel rod. A fuel rod is composed of a clad, a stack of pellets, and optionally a liner extruded on the inner surface of the clad. Each component of this architecture corresponds to a different block in the BISON input and mesh files. In the mesh input file, you refer to each block through a specific dictionary to create it. In the Exodus file, blocks are numbered, and a name is provided for each of them.

The pellets contained in a fuel rod can have different geometries. There is a block for each geometry, in the input file as well as in the Exodus file.

26.2 Input File Review

26.2.1 Pellet Type

This dictionary encapsulates a pellet geometry and the quantity of the corresponding pellets. To refer to a parameter, you have to know its key (the quoted string between brackets).

LINER
<u>INPUT FILE</u>
Dictionary: clad
Creation: clad['with_liner'] = True

<u>EXODUS FILE</u>
Type: block
Name: "liner"
Number: 2

CLAD
<u>INPUT FILE</u>
Dictionary: clad
Creation: automatic

<u>EXODUS FILE</u>
Type: block
Name: "clad"
Number: 1

PELLET TYPE #N
<u>INPUT FILE</u>
Dictionary: pellet_type_N
Creation: in list "pellets"

<u>EXODUS FILE</u>
Type: block
Name: "pellet_type_N"
Number: N+2

PELLET TYPE 1
<u>INPUT FILE</u>
Dictionary: pellet_type_1
Creation: in list "pellets"

<u>EXODUS FILE</u>
Type: block
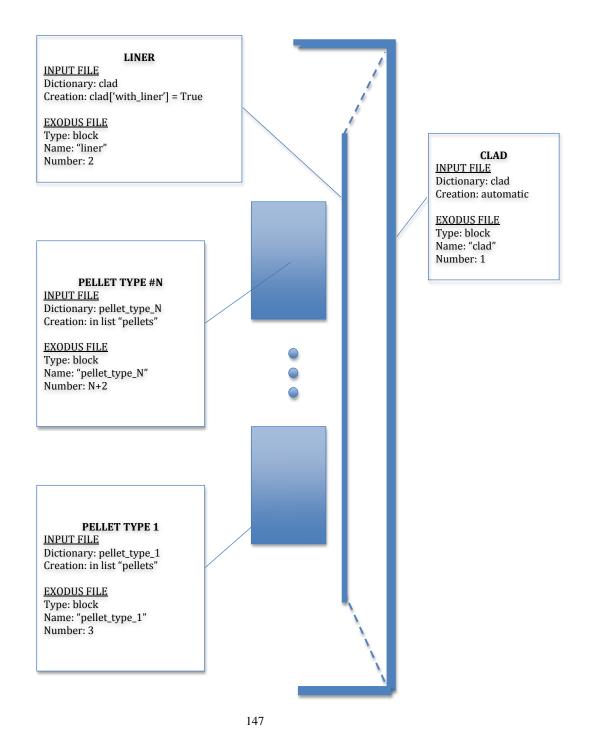Name: "pellet_type_1"
Number: 3

147

Figure 26.1: Overview of the architecture of a fuel rod.

```
# Pellet Type 1
Pellet1= {}
Pellet1['type'] = 'discrete'
Pellet1['quantity'] = 5
Pellet1['mesh_density'] = 'medium'
Pellet1['outer_radius'] = 0.0041
Pellet1['inner_radius'] = 0
Pellet1['height'] = 2*5.93e-3
Pellet1['dish_spherical_radius'] = 1.01542e-2
Pellet1['dish_depth'] = 3e-4
Pellet1['chamfer_width'] = 5.0e-4
Pellet1['chamfer_height'] = 1.6e-4
```

- `'type'` Type *string*. Must be 'discrete' or 'smeared'. From a geometric point of view, a smeared pellet is a rectangle. Two consecutive smeared pellets have their top and bottom surfaces merged.

- `'quantity'` Type *int*. Number of pellets created with this geometry.

- `'mesh_density'` Type *string*.

- `'outer_radius'` Type *float*. Outer radius of the pellet.

- `'inner_radius'` Type *float*. Inner radius of the pellet.

- `'height'` Type *float*. Pellet height.

- `'dish_spherical_radius'` Type *float*. Spherical radius of the dishing. Needed only if type is 'discrete'.

- `'dish_depth'` Type *float*. Depth of the dishing. Needed only if type is 'discrete'.

- `'chamfer_width'` Type *float*. Radial chamfer length in RZ coordinates. Must be zero for a non-chamfered pellet. Needed only if type is 'discrete'.

- `'chamfer_height'` Type *float*. Axial chamfer length in RZ coordinates. Must be zero for a non-chamfered pellet. Needed only if type is 'discrete'. If either `chamfer_width` or `chamfer_height` is zero, both must be zero.

26.2.2 Pellet Collection

```
pellets = [Pellet1, Pellet2, Pellet3]
```

This is a list of the pellets that make up the pellet stack. The geometries are ordered from the bottom to the top of the stack. A pellet type block must be present in this list to be created.

26.2.3 Stack Options

```
# Stack options
pellet_stack = {}
pellet_stack['interface_merge'] = 'point'
pellet_stack['higher_order'] = False
pellet_stack['angle'] = 0
```

- `'interface_merge'` Type *string*. Control type of merging between pellets. Options are: 'point' or 'no'. See Table 26.1 for a complete description. **Note that any other string results in pellets that are not merged.**

- `'higher_order'` Type *boolean*. Control order of mesh elements. See Table 26.2

- `'angle'` Type *int*. Between 0 and 360. Angle of revolution of the pellet stack. If 0, creates a 2D fuel rod. If greater than 0, creates a 3D fuel rod.

	2D discrete	2D smeared	3D discrete
`'point'`	vertex	curve	curve
`'no'`	not merged	not merged	not merged

Table 26.1: Merging control. 'Vertex' means that the pellets are merged at their common vertex which is the closest from the centerline. In 2D, 'curve' means that the pellets are merged at their common curve. In 3D, 'curve' means that the pellets are merged at the curve generated by the corresponding merged vertex in 2D RZ geometry.

	False	True
2D	QUAD4	QUAD8
3D	HEX8	HEX27

Table 26.2: Order of generated elements

26.2.4 Clad

```
clad = {}
clad['mesh_density'] = 'medium'
clad['gap_width'] = 8e-5
clad['bot_gap_height'] = 1e-3
clad['top_gap_height'] = 1.67e-3
clad['clad_thickness'] = 5.6e-4
clad['top_bot_clad_height'] = 2.24e-3
clad['plenum_fuel_ratio'] = 0.045
```

```
clad['with_liner'] = False
clad['liner_width'] = 5e-5
```

- 'mesh_density' Type *string*. CAUTION: the mesh density of the clad is related to the mesh density of the pellets which use the *same* mesh dictionary as the clad.

- 'gap_width' Type *float*. Radial width of the gap between the fuel and the clad (or the liner).

- 'bot_gap_height' Type *float*. Axial gap height between bottom of fuel and the cladding.

- 'top_gap_height' Type *float*. Axial gap height between top of fuel and the cladding. Either this or 'plenum_fuel ratio' must be given.

- 'clad_thickness' Type *float*. Thickness of the sleeve of the clad.

- 'top_bot_clad_height' Type *float*. Height of the bottom and of the top of the clad.

- 'plenum_fuel_ratio' Type *float*. Ratio of the axial gas height to the fuel height inside the cladding. Either this or 'top_gap_height' must be given.

- 'with_liner' Type *boolean*. Whether to include a liner.

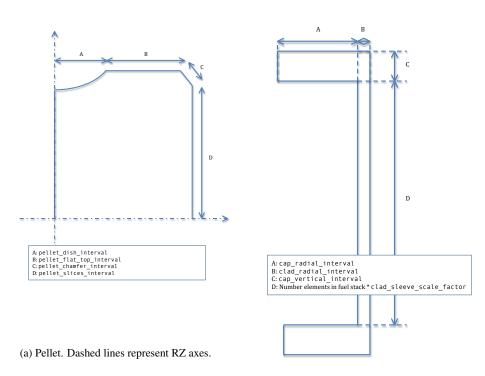- 'liner_width' Type *float*. Liner width.

26.2.5 Meshing Parameters

```
# Parameters of mesh density 'coarse'
coarse = {}
coarse['pellet_r_interval'] = 6
coarse['pellet_z_interval'] = 2
coarse['pellet_dish_interval'] = 3
coarse['pellet_flat_top_interval'] = 2
coarse['pellet_chamfer_interval'] = 1
coarse['pellet_slices_interval'] = 4
coarse['clad_radial_interval'] = 3
coarse['clad_sleeve_scale_factor'] = 4
coarse['cap_radial_interval'] = 6
coarse['cap_vertical_interval'] = 3
coarse['pellet_angular_interval'] = 6
coarse['clad_angular_interval'] =12
```

The user defines a dictionary containing the mesh parameters. The user can specify the name of this dictionary as long as the name is consistent with the names defined in the pellet type blocks for mesh_density. pellet_r_interval and pellet_z_interval are used only with smeared pellet meshes. Figure 26.2 explains other parameters.

The angular intervals are for 3D geometries and correspond to the created arcs of circle. Note that to have a nice mesh, you may want to have the same number of interval on the diameter of the fuel rod and on this arc of circle.

Figure 26.2: Mesh parameters

A: pellet_dish_interval
B: pellet_flat_top_interval
C: pellet_chamfer_interval
D: pellet_slices_interval

(a) Pellet. Dashed lines represent RZ axes.

A: cap_radial_interval
B: clad_radial_interval
C: cap_vertical_interval
D: Number elements in fuel stack * clad_sleeve_scale_factor

(b) Clad. Represented in RZ.

26.3 Output File Review

Figure 26.1 summarizes names and number of the blocks in the exodus file. Figure 26.4 summarizes the numbering for the sidesets and nodesets.
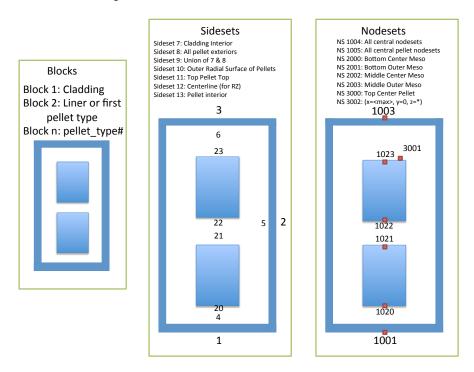

Figure 26.4: Sidesets, nodesets and blocks ids in the exodus file

26.4 Things to Know

26.4.1 Main Script

The main script is written in python v2.5. It is organized in classes: Pellet, PelletStack, Clad, Liner and FuelRod. The link between the input file and the main is assured by three functions. A first function is charged to pick read the input file. A second function checks that the syntax of the input file makes sense for the main script. The third function creates the mesh based on the input file.

26.4.2 Error Messages

AttributeError   Caused by a missing class in the input file.

KeyError   Often is caused by a wrong key in the input file. The main script should check that the keys entered in the input file are valid and specify which key is not valid if it occurs.

Other errors should be accompanied by a descriptive message. Contact the developers if the error message is not helpful.

Bibliography

[1] R. L. Williamson, J. D. Hales, S. R. Novascone, M. R. Tonks, D. R. Gaston, C. J. Permann, D. Andrs, and R. C. Martineau. Multidimensional multiphysics simulation of nuclear fuel behavior. *J. Nucl. Mater.*, 423:149–163, 2012.

[2] J. D. Hales, R. L. Williamson, S. R. Novascone, D. M. Perez, B. W. Spencer, and G. Pastore. Multidimensional multiphysics simulation of TRISO particle fuel. *J. Nucl. Mater.*, 443:531–543, November 2013.

[3] Pavel Medvedev. Fuel performance modeling results for representative FCRD irradiation experiments: Projected deformation in the annular AFC-3A U-10Zr fuel pins and comparison to alternative designs. Technical Report INL/EXT-12-27183 Revision 1, Idaho National Laboratory, 2012.

[4] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandié. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. *Nucl. Eng. Design*, 239:1768–1778, 2009.

[5] L. Schoof and V. Yarberry. EXODUS II: A finite element data model. Technical Report SAND92-2137, Sandia National Laboratories, September 1996.

[6] Sandia National Laboratories. CUBIT: Geometry and mesh generation toolkit. http://cubit.sandia.gov, 2008.

[7] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *J. Comput. Phys.*, 193(2):357–397, 2004.

[8] M. Kato, K. Maeda, T. Ozawa, M. Kashimura, and Y. Kihara. Physical properties and irradiation behavior analysis of Np- and Am-bearing MOX fuels. *Journal of Nuclear Science and Technology*, 48:646–653, 2011.

[9] C. M. Allison, G. A. Berna, R. Chambers, E. W. Coryell, K. L. Davis, D. L. Hagrman, D. T. Hagrman, N. L. Hampton, J. K. Hohorst, R. E. Mason, M. L. McComas, K. A. McNeil, R. L. Miller, C. S. Olsen, G. A. Reymann, and L. J. Siefken. SCDAP/RELAP5/MOD3.1 code manual, volume IV: MATPRO–A library of materials properties for light-water-reactor accident analysis. Technical Report NUREG/CR-6150, EGG-2720, Idaho National Engineering Laboratory, 1993.

[10] M. Kato, Y. Ikusawa, T. Sunaoshi, A. Nelson, and K. McClellan. Thermal expansion measurement of $(U, Pu)O_{2-x}$ in oxygen partial pressure-controlled atmosphere. *Journal of Nuclear Materials*, 469:223–227, 2016.

[11] M.V. Speight. A calculation on the migration of fission gas in material exhibiting precipitation and re-solution of gas atoms under irradiation. *Nuclear Science and Engineering*, 37:180–185, 1969.

[12] K. Lassmann and H. Benk. Numerical algorithms for intragranular fission gas release. *Journal of Nuclear Materials*, 280:127–135, 2000.

[13] J. A. Turnbull, R.J. White, and C. Wise. The diffusion coefficient for fission gas atoms in uranium dioxide. In *Proceedings of Technical Committee Meeting on Water Reactor Fuel Element Computer Modelling in Steady State, Transient and Accident Conditions*, pages 174–181, Preston, UK, Sept. 18–22, 1988.

[14] D. A. Andersson, P. Garcia, X. -Y. Liu, G. Pastore, M. Tonks, P. Millett, B. Dorado, D. R. Gaston, D. Andrs, R. L. Williamson, R. C. Martineau, B. P. Uberuaga, and C. R. Stanek. Atomistic modeling of intrinsic and radiation-enhanced fission gas (Xe) diffusion in $UO_{2\pm x}$: Implications for nuclear fuel performance modeling. *Journal of Nuclear Materials*, 451:225–242, 2014.

[15] J. A. Turnbull, C. A. Friskney, J. R. Findlay, F. A. Johnson, and A. J. Walter. The diffusion coefficients of gaseous and volatile species during the irradiation of uranium dioxide. *J. Nucl. Mater.*, 107:168–184, 1982.

[16] R.J. White and M.O. Tucker. A new fission-gas release model. *J. Nucl. Mater.*, 118:1–38, 1983.

[17] P. Lösönen. Modelling intragranular fission gas release in irradiation of sintered LWR UO_2 fuel. *Journal of Nuclear Materials*, 304:29–49, 2002.

[18] G. Pastore, L.P. Swiler, J.D. Hales, S.R. Novascone, D.M. Perez, B.W. Spencer, L. Luzzi, P. Van Uffelen, and R.L. Williamson. Uncertainty and sensitivity analysis of fission gas behavior in engineering-scale fuel modeling. *Journal of Nuclear Materials*, 465:398–408, 2015.

[19] F.S. Ham. Theory of diffusion-limited precipitation. *J. Phys. Chem. Solids*, 6:335–351, 1958.

[20] D. R. Olander and D. Wongsawaeng. Re-solution of fission gas – A review: Part I. Intragranular bubbles. *Journal of Nuclear Materials*, 354:94–109, 2006.

[21] P. Hermansonn and A.R. Massih. An effective method for calculation of diffusive flow in spherical grains. *Journal of Nuclear Materials*, 304:204–211, 2002.

[22] D. Pizzocri, C. Rabiti, L. Luzzi, T. Barani, P. Van Uffelen, and G. Pastore. PolyPole-1: An accurate numeical algorithm for intra-granular fission gas release. *Journal of Nuclear Materials*, 478:333–342, 2016.