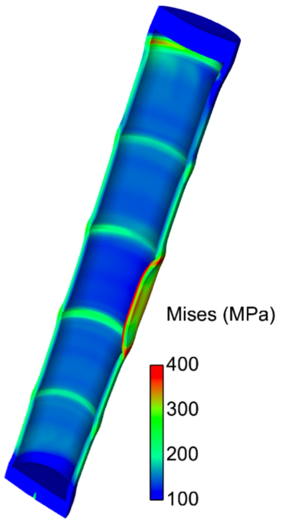
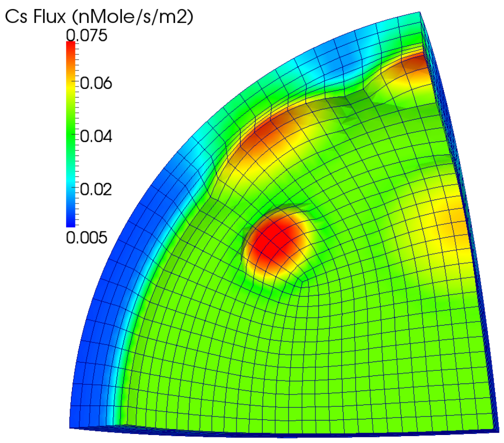
BISON Users Manual

October 2013





Fuels Modeling and Simulation Department
Idaho National Laboratory

BISON Users Manual

J. D. Hales, S. R. Novascone, G. Pastore,
D. M. Perez, B. W. Spencer, R. L. Williamson

Fuels Modeling & Simulation Department
Idaho National Laboratory
Idaho Falls, ID

October 2013

Contents

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 22.

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

To checkout the code (for INL onsite users):

```
cd ~/projects
svn co https://hpcsc.inl.gov/svn/herd/trunk
```

For offsite users:

```
cd ~/projects
svn co https://localhost:4443/svn/herd/trunk
```

It is necessary to build libmesh before building any application.

```
cd ~/projects/trunk/libmesh
./build_libmesh_moose.sh
```

Once libmesh has compiled successfully, you may now compile BISON.

```
cd ~/projects/trunk/bison/
make (add -jn to run on multiple "n" processors)
```

Once BISON has compiled successfully, it is recommended to run the tests to make sure the version of the code you have is running correctly.

```
cd ~/projects/trunk/bison/
./run_test (add -jn to run "n" jobs at one time)
```

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/trunk/bison/examples/2D-RZ_rodlet_10pellets
# To run with one processor
~/projects/trunk/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 `PiecewiseLinearFile` function is used 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 = PiecewiselinearFile
  yourFileName = 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 22, 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_height = 1.0e-3
   pellet_quantity = 10
   pellet_height = 0.01186
   plenum_fuel_ratio = 0.045
   clad_gap_width = 8e-5
   to_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 and can be accessed by running `../peacock/peacock` from the BISON directory. Information about Peacock 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.

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 is available on the MOOSE wiki page. See the link for Input File Syntax.

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, 15, 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 17 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 16 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.

4 Global Parameters

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

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 above, the line commands `order = FIRST` and `family = LAGRANGE` will be available in all blocks and subblocks in the remainder of the input file.

5 Problem

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

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.

6 Mesh

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

`file`	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. A typical value is 1000.

The `Mesh` block's purpose is to give details about the finite element mesh to be used.

7 Variables

```
[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.

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`).

8 AuxVariables

```
[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.

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.

9 Functions

9.1 ParsedFunction

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

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

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/`.

9.2 PiecewiseLinear

```
[./piecewiselinear]
  type = PiecewiseLinear
  x = <real list>
  y = <real list>
  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.
`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.

The `PiecewiseLinear` function takes pairs of x-y data as input and interpolates values based on those pairs. 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.

9.3 PiecewiseBilinear

```
[./piecewiselinear]
  type = PiecewiseBilinear
  yourFileName = <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`
`yourFileName`	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`.

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.

9.4 Composite

```
[./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.

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.

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 Dirichlet

10.1.1 DirichletBC

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

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

10.1.2 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.

10.1.3 FunctionDirichletBC

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

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

10.1.4 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.

10.2 Pressure

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

`boundary`	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.

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.

10.3 PlenumPressure

```
[./PlenumPressure]
  [./plenumpressure]
    boundary = <string list>
    initial_pressure = <real> (0)
    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`	List of boundary names or ids where this boundary condition will apply.
`initial_pressure`	The initial pressure in the plenum.
`startup_time`	The amount of time during which the pressure will ramp from zero to its true value.
`R`	The universal gas constant. In BISON, SI units are used, and R should be 8.3143.
`output_initial_moles`	If given, the reporting `Postprocessor` to use for the initial moles of gas.
`temperature`	The name of the `Postprocessor` holding the average temperature value.
`volume`	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 reporting `Postprocessor` to use for the plenum pressure value.
`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`.

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.

10.4 CoolantChannel

```
[CoolantChannel]
  [./coolantchannel]
    boundary = <string list>
    variable = <string>
    axial_power_profile = <string>
    cond_metal = <real>
    cond_oxide = <real>
    coupledEnthalpy = <string>
    direction = <string>
    direction2 = <string>
    flow_area = <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>
    inlet_massflux = <string or real>
    inlet_pressure = <string or real>
    inlet_temperature = <string or real>
    linear_heat_rate = <string>
    number_axial_zone = <integer> (0)
    number_lateral_zone = <integer> (1)
    oxide_thickness = <string>
    oxide_model = <string> (zirconia)
    pbr = <real>
    rod_diameter = <real> (0.01)
    rod_pitch = <real> (0.0126)
  [../]
[]
```

boundary	List of boundaries. Typically only one boundary id is given.
variable	Name of variable associated with this BC. Typically temp.
axial_power_profile	Function name for function describing axial power factors.
cond_metal	Conductivity of the metal. Used if oxide_model is user.
cond_oxide	Conductivity of the oxide. Used if oxide_model is user.

`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.
`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.
`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), or 5 (DNB low flow).
`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), 4 (Shrock-Grossman), or 5 (constant).
`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.
`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.
`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.
`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.
`rod_diameter`	Diameter of the fuel rod.
`rod_pitch`	Pitch or spacing between fuel rods.

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.

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.

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)
    slave = <string>
    tangential_tolerance = <real>
    tension_release = <real>
  [../]
[]
```

`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`.
`formulation`	One of DEFAULT or PENALTY.
`friction_coefficient`	The friction coefficient.
`master`	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.
`slave`	The boundary id for the slave surface.
`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. Must be positive.

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
    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>
    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>
    stefan_boltzmann = <real> (5.669e-8)
    variable = <string>
  [../]
[]
```

`type`	GapHeatTransfer
`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.
`master`	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`	The boundary id for the slave surface.
`stefan_boltzmann`	The Stefan-Boltzmann constant.
`tangential_tolerance`	Tangential distance to extend edges of contact surfaces.

The `quadrature` option is recommended with second-order meshes.

11.2.2 GapHeatTransferLWR

```
[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)
    slave = <string>
    stefan_boltzmann = <real> (5.669e-8)
    variable = <string>
  [../]
[]
```

`type`	GapHeatTransferLWR
`contact_coef`	The leading coefficient on the solid-solid conduction relation ($1/\sqrt{\text{m}}$).
`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.
slave	The boundary id for the slave surface.
stefan_boltzmann	The Stefan-Boltzmann constant.
tangential_tolerance	Tangential distance to extend edges of contact surfaces.

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 (10.3).)

The quadrature option is recommended with second-order meshes.

12 AuxKernels and AuxBCs

`AuxKernels` and `AuxBCs` are used to compute values for `AuxVariables`. They often compute quantities based on functions, solution variables, and material properties.

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

[AuxBCs]
  [./name]
    type = <AuxBC type>
    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.

All `AuxKernels` act on blocks. All `AuxBCs` act on boundaries. If no block or boundary is specified, the `AuxKernel` or `AuxBC` will act on the entire model.

Note that the same types are recognized in `AuxKernels` and `AuxBCs`.

12.1 AuxKernels for Output

12.1.1 MaterialTensorAux

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

```
[../]
```

`type`	`MaterialTensorAux`
`tensor`	Name of second-order tensor material property. A typical second-order tensor material property is `stress`.
`variable`	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.

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. The computed value will be the volume-averaged quantity over the element.

12.1.2 MaterialRealAux

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

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

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.

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 FissionRateAuxLWR

```
[./fissionrateauxlwr}
  type = FissionRateAuxLWR
  value = <real> (1)
  function1 = <string>
  function2 = <string>
  pellet_diameter = <real> (0.0082)
  pellet_inner_diameter = <real> (0)
  fuel_volume_ratio = <real> (1)
  energy_per_fission = <real> (3.28451e-11)
[../]
```

`value`	Fission rate if `function1` is not present. Scale factor if `function1` is given.
`function1`	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.
`function2`	Function describing axial power profile.
`pellet_diameter`	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.

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

12.2.2 FissionRateAux

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

`type`	`FissionRateAux`
`variable`	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.

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.

12.2.3 FissionRateFromPowerDensity

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

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

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).

12.3 Other AuxKernels

12.3.1 Al2O3Aux

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

`type`	`Al2O3Aux`
`variable`	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]
  type = BurnupAux
  fission_rate = <string>
  density = <real>
  molecular_weight = <real> (0.270)
[../]
```

`type`	`BurnupAux`
`variable`	Variable name corresponding to the burnup. Typically `burnup`.
`fission_rate`	Variable name corresponding to the fission rate. Typically `fission_rate`.
`density`	The initial fuel density.
`molecular_weight`	The molecular weight.

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

12.3.3 FastNeutronFluxAux

```
[./fastneutronfluxaux]
  type = FastNeutronFluxAux
  variable = <string>
  fast_neutron_flux = <string>
[../]
```

`type`	`FastNeutronFluxAux`
`variable`	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 `rod_ave_lin_pow` is not given. Otherwise, a scale factor. Recommended scale factor value is 3e13 (n/(m^2-s)/(W/m)).

12.3.4 FastNeutronFluenceAux

```
[./fastneutronfluenceaux]
  type = FastNeutronFluenceAux
  variable = <string>
  fast_neutron_flux = <string>
[../]
```

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

12.3.5 GrainRadiusAux

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

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

The `GrainRadiusAux` model is a simple empirical model for calculating grain growth. This can be used with the Sifgrs model (15.3.2).

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)
  temp = <string>
  use_coolant_channel = <bool> (false)
```

`type`	`OxideAux`
`variable`	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%.
temp	Variable name for temperature variable. Typically temp.
use_coolant_model	If true, model will adjust surface temperature based on the coolant channel model.

12.3.7 PelletIdAux

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

type	PelletIdAux
a_lower	The lower axial coordinate of the fuel stack.
a_upper	The upper axial coordinate of the fuel stack.
number_pellets	Number of fuel pellets.

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

13 Burnup

```
[Burnup]
  [./burnup]
    rod_ave_linear_power = <string>
    axial_power_profile = <string>
    num_radial = <integer>
    num_axial = <integer>
    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)
    i_enrich = <real list> (0.05, 0.95, 0, 0, 0, 0)
    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> (sum of sigma_c and sigma_f)
    N235 = <string>
    N238 = <string>
    N238 = <string>
    N240 = <string>
    N241 = <string>
    N242 = <string>
    RPF = <string>
  [../]
[]
```

`block`	List of fuel blocks. Either block numbers or names.
`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.
`a_lower`	The lower axial coordinate of the fuel stack.
`a_upper`	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`	The initial fuel density.
`energy_per_fission`	The energy released per fission in J/fission.
`i_enrich`	The initial enrichment for the six isotopes.
`sigma_c`	The capture cross sections for the six isotopes.
`sigma_f`	The fission cross sections for the six isotopes.
`sigma_a_thermal`	The absorption (thermal) cross sections for the six isotopes.
`N235`	Indicates that the output of the concentration of N235 is required. Typically N235.
`N238`	Indicates that the output of the concentration of N238 is required. Typically N238.
`N239`	Indicates that the output of the concentration of N239 is required. Typically N239.
`N240`	Indicates that the output of the concentration of N240 is required. Typically N240.
`N241`	Indicates that the output of the concentration of N241 is required. Typically N241.
`N242`	Indicates that the output of the concentration of N242 is required. Typically N242.
`RPF`	Indicates that the output of the radial power factor is required. Typically RPF.

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.

14 Kernels

Kernels are used to volume integrals associated with a given term in a PDE. They often compute quantities based on functions, solution variables, auxiliary variables, and material properties.

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

type	Type of kernel.
block	List of blocks. Either block numbers or names.

All Kernels act on blocks. If no block is specified, the Kernel will act on the entire model.

14.1 SolidMechanics

```
[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.

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.

14.2 Gravity

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

`type`	`Gravity`
`variable`	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).

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

14.3 Heat Conduction

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

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

Kernel for diffusion of heat or divergence of heat flux.

14.4 Heat Conduction Time Derivative

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

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

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

14.5 Neutron Heat Source

```
[./neutronheatsource]
  type = NeutronHeatSource
  variable = <variable>
  fission_rate = <variable>
[../]
```

type	NeutronHeatSource
variable	Variable name corresponding to the heat conduction equation. Typically temp.
fission_rate	Variable name corresponding to the fission rate. Typically fission_rate.

Kernel for the volumetric heat source associated with fission.

14.6 BodyForce

```
[./bodyforce]
  type = BodyForce
  variable = <variable>
  value = <real>
  function = <string>
[../]
```

type	BodyForce
variable	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.

Kernel for applying an arbitrary body force to the model.

14.7 TimeDerivative

```
[./timederivative]
```

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

`type`	`TimeDerivative`
`variable`	Variable associated with this volume integral.

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

14.8 Arrhenius Diffusion

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

`type`	`ArrheniusDiffusion`
`variable`	Variable associated with this volume integral.

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

15 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.

15.1 Thermal Models

15.1.1 HeatConductionMaterial

```
  [./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.

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

15.1.2 ThermalFuel

```
[./thermalfuel]
  type = ThermalFuel
  temp = <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	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.

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

15.2 Solid Mechanics Models

15.2.1 CreepPyC

```
[./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	Variable name corresponding to the fast neutron flux. Typically fast_neutron_flux.
density	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.

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

15.2.2 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`	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

$$\dot{\varepsilon}_{cr} = K\sigma\phi. \tag{15.1}$$

15.2.3 CreepUO2

```
  [./creepuo2]
    type = CreepUO2
    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>
  [../]
```

`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`.
`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 [8].
matpro_poissons_ratio	Set to true to use correlations for Poisson's modulus from MATPRO [8].
matpro_thermal_expansion	Set to true to use correlations for coefficient of thermal expansion from MATPRO [8].
burnup	Name of burnup variable. Only required if using MATPRO correlations. Typically `burnup`.

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

15.2.4 Elastic

```
[./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.

The Elastic model is a simple hypo-elastic model.

15.2.5 IrradiationGrowthZr4

```
[./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.

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

15.2.6 PyCIrradiationStrain

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

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

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.

15.2.7 MechZry

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

```
    initial_fast_fluence = <real> (0.0)
    cold_work_factor = <real> (0.01)
    oxygen_concentration = <real> (0.0)
    relative_tolerance = <real> (1e-4)
    absolute_tolerance = <real> (1e-20)
    max_its = <integer> (10)
    output_iteration_info = <bool> (false)
    model_irradiation_growth = <bool> (true)
    model_primary_creep = <bool> (true)
    model_thermal_creep = <bool> (true)
    model_irradiation_growth = <bool> (true)
    model_thermal_expansion = <bool> (true)
    model_elastic_modulus = <bool> (false)
    stress_free_temperature = <real>
    material_type = < 0 or 1 for SRA or RXA >
  [../]
```

`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.
`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.
`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_irradiation_growth`	Whether to model irradiation growth.
`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. 0 for SRA, 1 for RXA.

The `MechZry` model includes the option to model primary, thermal, and irradiation-induced creep. It is also possible to turn on irradiation growth. If irradiation growth is turned on, do not include the `IrradiationGrowthZr4` model.

15.2.8 RelocationUO2

```
[./relocationuo2]
  type = RelocationUO2
  burnup = <string>
  diameter = <real>
  q = <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>
[../]
```

`type`	`RelocationUO2`
`burnup`	Name of burnup variable. Typically `burnup`.
`diameter`	As fabricated cold diameter of pellet in meters.
`q`	Linear heat rate in pellet in W/m.
`gap`	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.

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.

15.2.9 ThermalIrradiationCreepZr4

```
[./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.

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

15.2.10 VSwellingUO2

```
[./vswellinguo2]
  type = VSwellingUO2
  temp = <string>
  burnup = <string>
  density = <real>
  total_densification = <real> (0.01)
  complete_burnup = <real> (5)
[../]
```

type	VSwellingUO2
temp	Name of temperature variable. Typically temp.
burnup	Name of burnup variable. Typically burnup.
density	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).

The VSwellingUO2 model computes a volumetric strain to account for solid and gaseous swelling and for densification.

15.3 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.

15.3.1 ForMas

```
[./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)
  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.

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

15.3.2 Sifgrs

```
[./sifgrs]
  type = Sifgrs
  initial_grain_radius = <real> (5e-6)
```

```
   hydrostatic_stress_const = <real> (0.0)
   surface_tension = <real> (0.5)
   saturation_coverage = <real> (0.5)
   hbs_release_burnup = <real> (100)
   initial_porosity = <real> (0.05)
   density = <real>
   solid_swelling_factor = <real> (5.577e-5)
   total_densification = <real> (0.01)
   end_densification_burnup = <real> (5)
   pellet_brittle_zone = <string>
   diff_coeff_option <integer>
   compute_swelling = <bool> (false)
   ath_model = <bool> (false)
   gbs_model = <bool> (false)
   ramp_model = <bool> (false)
   hbs_model = <bool> (false)
   file_name = <string>
   format = <string> (rows)
   rod_ave_lin_power = <string>
   axial_power_profile = <string>
   grain_radius = <string>
   pellet_id = <string>
   temp = <string>
   fission_rate = <string>
   hydrostatic_stress = <string>
   burnup = <string>
 [../]
```

`type`	`Sifgrs`
`initial_grain_radius`	Initial grain radius.
`hydrostatic_stress_const`	A constant value for hydrostatic stress. Ignored if `hydrostatic_stress` is given.
`surface_tension`	Bubble surface tension (J/m^2).
`saturation_coverage`	Fractional grain boundary bubble coverage at saturation.
`hbs_release_burnup`	Threshold local burnup for gas release from the HBS porosity (MWd/kgU).
`initial_porosity`	Initial fuel porosity.
`density`	Initial fuel density.
`solid_swelling_factor`	Solid swelling coefficient.
`total_densification`	The densification that will occur given as a fraction of theoretical density.
`end_densification_burnup`	The burnup at which densification is complete (MWD/kgU).
`pellet_brittle_zone`	The name of the `UserObject` that computes the width of the brittle zone.

diff_coeff_option	One of 0 (Turnbull), 1 (Andersson, low burnup), 2 (Andersson, high burnup), or 3 (Turnbull modified).
compute_swelling	Whether to compute fuel swelling.
ath_model	Whether to compute athermal gas release.
gbs_model	Whether to compute grain boundary sweeping.
ramp_model	Whether to include the ramp release model. Requires file_name.
hbs_model	Whether to include high burnup structure gas release.
file_name	File 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.
format	One of rows or columns.
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.
grain_radius	Variable name for grain radius.
pellet_id	Variable name for pellet id. Typically pellet_id.
temp	Variable name for temperature variable. Typically temp.
fission_rate	Variable name corresponding to the fission rate. Typically fission_rate.
hydrostatic_stress	Variable name for hydrostatic stress. Typically hydrostatic_stress.
burnup	Name of burnup variable. Typically burnup.

Sifgrs is the recommended fission gas model.

15.4 Mass Diffusion Models

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

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

$$d = d_1 \exp\left(\frac{-q_1}{RT}\right) + d_2 \exp\left(\frac{-q_2}{RT}\right). \tag{15.2}$$

15.5 Other Models

15.5.1 Density

```
[./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	Density.

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

16 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>
    output = <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.
`output`	The options are: none, screen, file, both, auto (no output, output to screen only, output to files only, output both to screen and files, same as both but no warnings if output options conflict).

All `Postprocessors` act on either boundaries or blocks. If no block or boundary is specified, the `Postprocessor` will act on the entire model.

16.1 SideAverageValue

```
  [./sideaveragevalue}
    type = SideAverageValue
    variable = <string>
  [../]
```

`type`	`SideAverageValue`
`variable`	The variable this `Postprocessor` acts on.

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.

16.2 InternalVolume

```
[./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.

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.

16.3 Reporter

```
[./reporter]
  type = Reporter
  default = <real> (0)
[../]
```

type	Reporter
default	Default or initial value of the Postprocessor.

Reporter is a unique Postprocessor in that it does not calculate anything at all. It is simply a scalar value that can be set and used by other MOOSE objects. It is commonly used to report scalar quantities computed by boundary conditions, kernels, and other objects.

16.4 TimestepSize

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

type	TimestepSize

TimestepSize reports the timestep size.

16.5 NumNonlinearIterations

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

type	NumNonlinearIterations

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

16.6 PlotFunction

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

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

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

16.7 ElementIntegralPower

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

type	ElementIntegralPower
fission_rate	Variable name corresponding to the fission rate. Typically fission_rate.

`energy_per_fission`	The energy released per fission in J/fission.

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

16.8 SideFluxIntegral

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

`type`	`SideFluxIntegral`
`variable`	Variable to be used in the flux calculation.
`diffusivity`	The diffusivity material property to be used in the calculation.

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

17 Executioner

The `Executioner` block describes how the simulation will be executed. It includes commands to control the solver behavior and time stepping.

```
[Executioner]
  type = <string>
  solve_type = <string>
  print_linear_residuals = <bool> (false)
  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>
  optimal_iterations = <integer>
  iteration_window = <integer> (0.2*optimal_iterations)
  linear_iteration_ratio = <integer> (25)
```

`type`	Several available. Typically `AdaptiveTransient`.
`solve_type`	One of `PJFNK` (preconditioned JFNK), `JFNK` (JFNK), `NEWTON` (Newton), or `SolveFD` (Jacobian computed by finite difference–serial only, slow).
`print_linear_residuals`	Whether to print linear residuals to the screen.
`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.
`dt`	The initial timestep size.
`end_time`	The end time of the analysis.
`num_steps`	The maximum number of time steps.
`dtmax`	The maximum allowed timestep size. Used with `AdaptiveTransient`.
`dtmin`	The minimum allowed timestep size. Used with `AdaptiveTransient`.
`optimal_iterations`	The target number of nonlinear iterations for adaptive timestepping. Used with `AdaptiveTransient`.
`iteration_window`	The size of the nonlinear iteration window for adaptive timestepping. Used with `AdaptiveTransient`.
`linear_iteration_ratio`	The ratio of linear to nonlinear iterations to determine target linear iterations and window for adaptive timestepping.

Many `Executioner` types exist. 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.

18 Output

```
[Output]
  file_base = <string> (mesh file base name + '_out')
  interval = <integer> (1)
  exodus = <bool> (false)
  max_pps_rows_screen = <integer> (15)
  postprocessor_csv = <bool> (false)
  output_initial = <bool> (false)
  [../]
```

`file_base`	Base file name for output files.
`interval`	The interval at which solutions are written to the output files.
`exodus`	Specifies that you would like ExodusII solution files. Typically set to true.
`max_pps_rows_screen`	The maximum number of postprocessor values displayed on screen during a timestep (set to 0 for unlimited).
`postprocessor_csv`	Specifies whether you would like a csv file containing `Postprocessor` values.
`output_initial`	Specifies whether you would like the initial state of the model written to the output file. Typically set to true.

The `Output` block lists parameters that control the frequency and type of results files produced.

19 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.

19.1 MaxIncrement

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

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

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

20 UserObjects

20.1 PelletBrittleZone

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

`type`	`PelletBrittleZone`
`pellet_id`	Variable name for pellet id. Typically `pellet_id`.
`temp`	Name of temperature variable. Typically `temp`.
`pellet_radius`	The outer radius of the fuel.
`a_lower`	The lower axial coordinate of the fuel stack.
`a_upper`	The upper axial coordinate of the fuel stack.
`number_pellets`	Number of fuel pellets.

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

21 Timestepping

The time steps taken by BISON can be specified directly by providing either a single fixed time step to take throughout the analysis, or by providing the time step as a function of time. Alternatively, an 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.

All of these types of timestepping can be obtained by using the `AdaptiveTransient` type of executioner. The parameters used in this executioner to obtain these different types of time stepping are described below.

21.1 Direct Time Step Control with Constant Time Step

The most basic way to control the time steps taken by BISON is to use the `AdaptiveTransient` executioner with options that instruct it to take a single, fixed time step over the duration of the analysis. To take time steps in this way, simply specify the time step to be taken using the `dt` parameter.

While using a constant time step, 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 the factor specified by the user through the `cutback_factor` parameter.

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 optionally specify a minimum time step through the `dtmin` parameter. 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 has been cut back to obtain a solution, BISON uses that cut-back time step in the next step. If that solution is successful, BISON attempts to increase the time step by multiplying it by the value specified by the `growth_factor` parameter. This is done repeatedly until the time step has grown back to the original value specified in the `dt` parameter.

21.2 Direct Time Step Control with Varying Time Step Size

BISON can optionally take time steps that are specified by the user, but which can vary over time. This is accomplished by providing a set of pairs of times and time steps instead of with a single fixed time step. 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 a fixed time step is used to cut back the time step from the user-specified value if a failed solution occurs. The time step is grown until it reaches to the value specified by the time-dependent function in the same way that is done with a fixed time step.

21.3 Adaptive Time Stepping

The two methods for user-specified time stepping described above can be used to cut the time step back if a solution fails. While this technique can be helpful to get past difficult parts of the time history, it can be much more efficient to adapt the time step based on the difficulty of the solution.

The `AdaptiveTransient` executioner provides an option to grow or shrink the time step based on the number of iterations taken to obtain a converged solution in the last converged step. The adaptive time stepping option is activated by setting a value for the `optimal_iterations` parameter. This parameter is the number of nonlinear iterations per time step that provides optimal solution efficiency. If more iterations than that are required, 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 21.1, 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 using the `dtmax` and `dtmin` parameters, respectively.

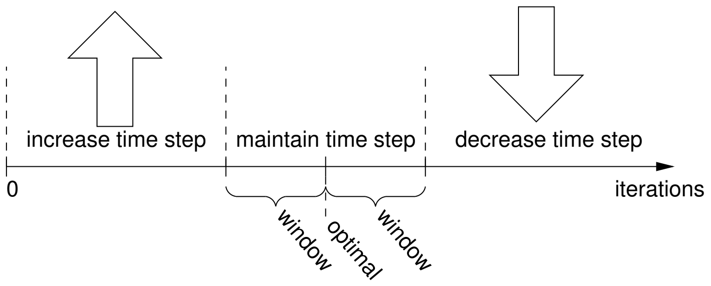


Figure 21.1: Criteria used to determine adaptive time step size

22 Mesh Script

22.1 Overview

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

22.1.1 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]
```

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. Finally, you may supply any mesh script input file with the `-i` flag.

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

22.1.2 Mesh Architecture

Figure 22.1 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.

22.2 Input File Review

22.2.1 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).

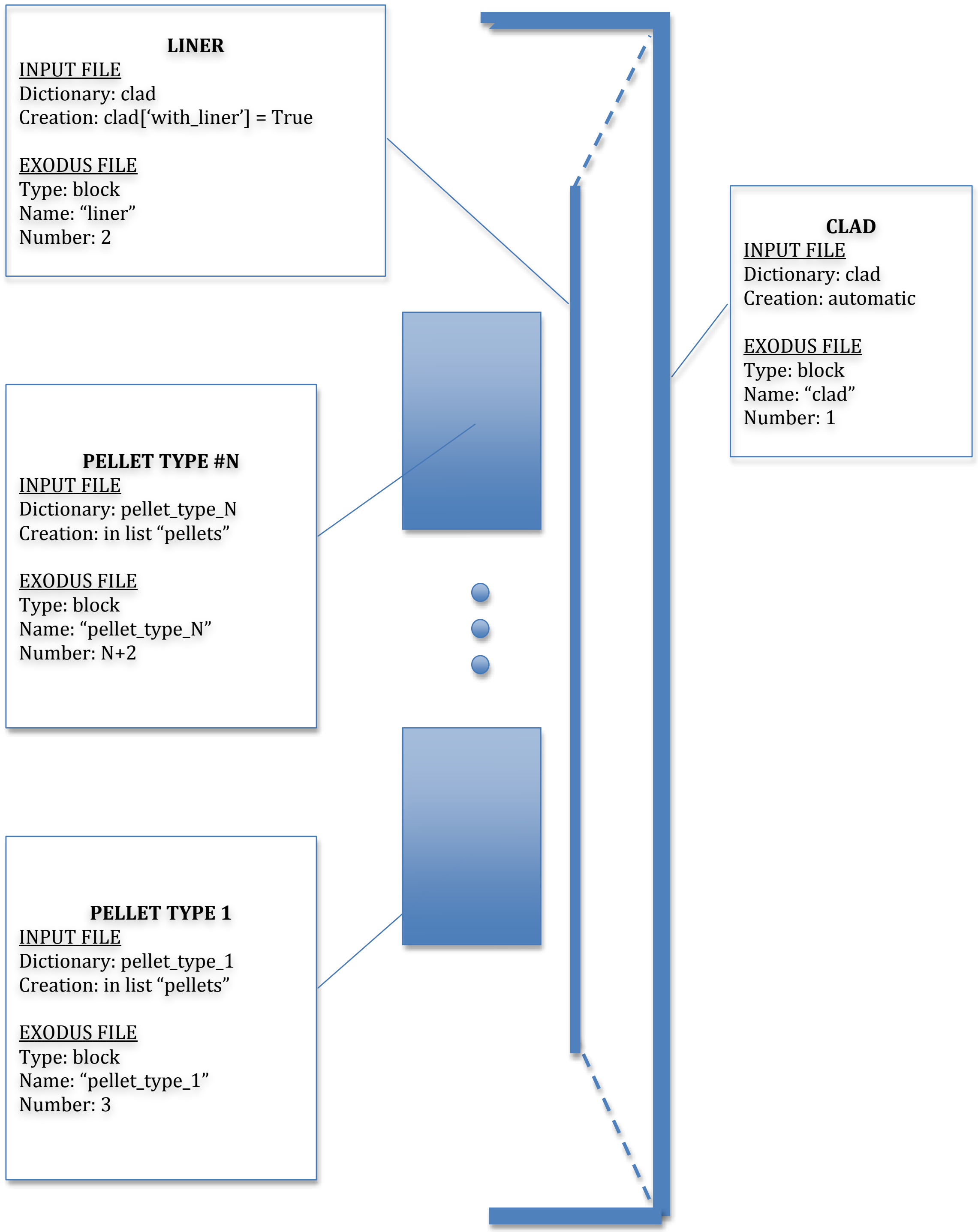


Figure 22.1: Overview of the architecture of a fuel rod.

```
# 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.

22.2.2 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.

22.2.3 Stack Options

```
# Stack options
pellet_stack = {}
pellet_stack['merge_pellets'] = True
pellet_stack['higher_order'] = False
pellet_stack['angle'] = 0
```

- `'merge_pellets'` Type *string*. Control type of merging between pellets. Options are: 'yes', 'no', 'point', 'surface'. See Table 22.1 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 22.2
- `'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**
`'yes'`	vertex	curve	curve
`'no'`	not merged	not merged	not merged
`'point'`	vertex	vertex	curve
`'surface'`	not merged	curve	not merged

Table 22.1: 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	HEX20

Table 22.2: Order of generated elements

22.2.4 Clad

```
clad = {}
clad['mesh_density'] = 'medium'
clad['gap_width'] = 8e-5
clad['bot_gap_height'] = 1e-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 height between fuel and top/bottom of the gap.
- `'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 free volume by the volume of the fuel.
- `'with_liner'` Type *boolean*. Whether to include a liner.
- `'liner_width'` Type *float*. Liner width.

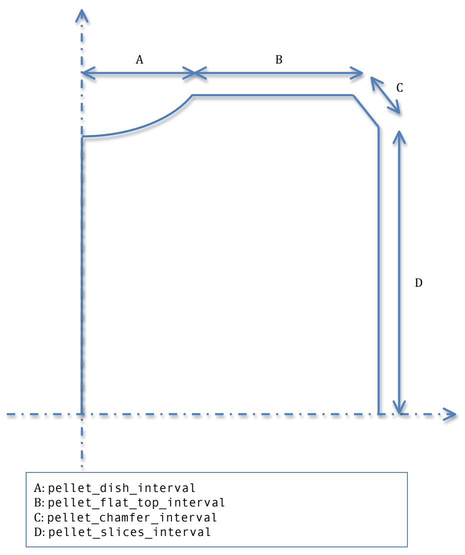
22.2.5 Meshing Parameters

```
# 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 22.2 explains other parameters.

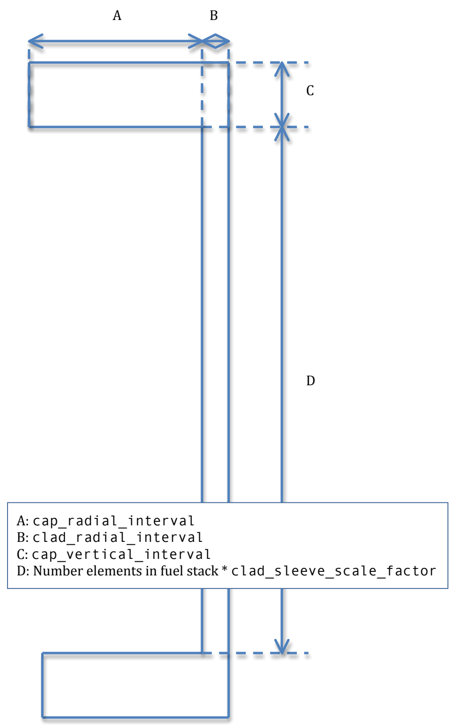
The angular intervals are for 3D geometries and correspond to the created arcs of circle. Note that to have a nice mesh, you may want to have the same number of interval on the diameter of the fuel rod and on this arc of circle.

Figure 22.2: Mesh parameters



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

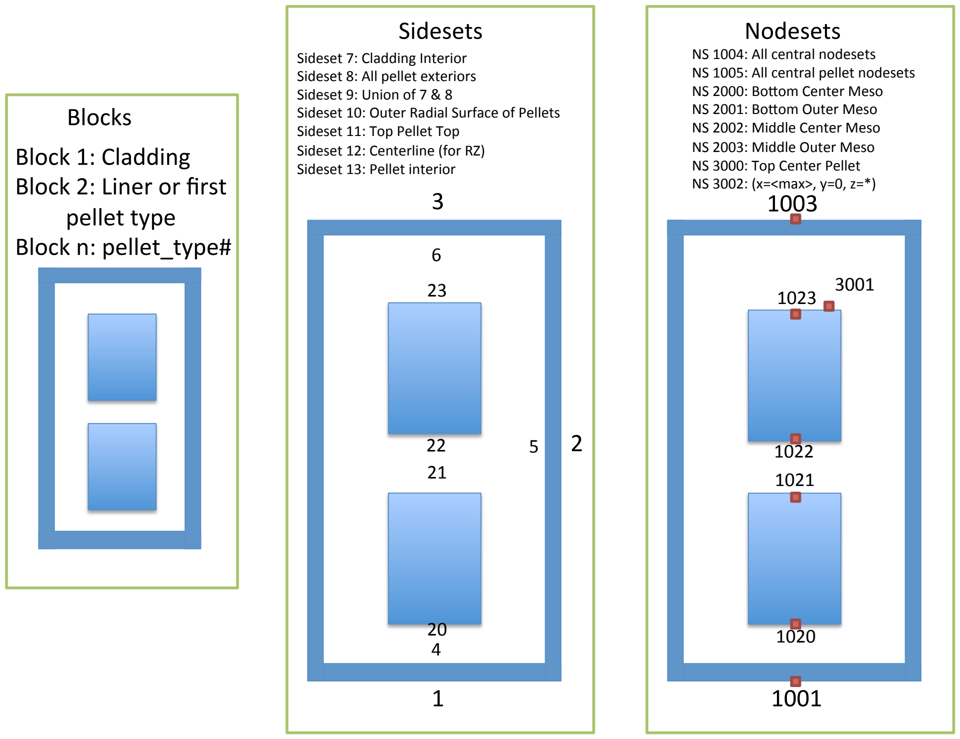
(a) Pellet. Dashed lines represent RZ axes.



(b) Clad. Represented in RZ.

22.3 Output File Review

Figure 22.1 summarizes names and number of the blocks in the exodus file. Figure 22.3 summarizes the numbering for the sidesets and nodesets.



Blocks
Block 1: Cladding
Block 2: Liner or first pellet type
Block n: pellet_type#
Sidesets
Sideset 7: Cladding Interior
Sideset 8: All pellet exteriors
Sideset 9: Union of 7 & 8
Sideset 10: Outer Radial Surface of Pellets
Sideset 11: Top Pellet Top
Sideset 12: Centerline (for RZ)
Sideset 13: Pellet interior
3
6
23
22
5
2
21
20
4
1
Nodesets
NS 1004: All central nodesets
NS 1005: All central pellet nodesets
NS 2000: Bottom Center Meso
NS 2001: Bottom Outer Meso
NS 2002: Middle Center Meso
NS 2003: Middle Outer Meso
NS 3000: Top Center Pellet
NS 3002: (x=<max>, y=0, z=*)
1003
1023
3001
1022
1021
1020
1001

Figure 22.3: Sidesets, nodesets and blocks ids in the exodus file

22.4 Things to Know

22.4.1 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.

22.4.2 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. Nuclear Materials*, 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. Nuclear Materials*, 443:531–543, 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] 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.