TEM3P Top-Level Commands

- **ThermostaticProblem**
- **ElasticProblem**
- **ThermoElasticProblem**

Note: Refer to **acdtool** command syntax for postprocessing capabilities.

TEM3P Input Specifications

- Multiphysics Analysis
- Input File
- Solver
- Material Property
- Mechanical Analysis Boundary
- Thermal Analysis Boundary
- Multiphyscis Boundary
- Lossy Dielectric Heat
- External Volume Heat
- Thermal Shell
- Nonlinear Material Property
- Mesh Deformation

Multiphysics Analysis Specification

ThermostaticProblem

- Steady state thermal analysis
- Linear and nonlinear analysis
- Coupled from electromagnetic fields

ElasticProblem

- Structural analysis
- Linear elasticity analysis
- Coupled from electromagnetic fields
- Eigenmode analysis
- Harmonic response analysis

TheromoElasticProblem

- Combined thermal and structural analysis
- Coupled from electromagnetic fields for thermal analysis
- Coupled from thermal solution for structural analysis

Each analysis type is specified by a problem type declaration, followed by a pair of brackets. Detailed information of each analysis type including material properties, boundary conditions and solvers is included in these brackets. For combined thermal and structural analyses, both thermal and elastic analysis specifications need to be included, preceded by **ThermoElasticProblem** definition. Their syntaxes are shown below.

Thermostatic problem specification

```
ThermostaticProblem: {
  . . .
}
```

Elasitc problem specification

```
ElasticProblem: {
  . . .
}
```

Thermo-Elastic problem specification

```
ThermoElasticProblem: {
   RunId: 3
}

ThermostaticProblem: {
  . . .
}

ElasticProblem: {
  . . .
}
```

Input File Specification

```
ThermostaticProblem: {

  MeshFile: RfGunBody.ncdf
  BasisOrder: 1
  CurvedSurfaces: on

}

ElasticProblem: {

  MeshFile: RfGunBody.ncdf
  BasisOrder: 1
  CurvedSurfaces: on

}
```

Both **ThermostaticProblem** and **ElasticProblem** have the same convention for input files. **ThermostaticProblem** defines the problem as thermal static problem. **ElasticProblem** defines the problem as structural analysis problem

MeshFile: The name of the mesh file in netcdf format. It can be located in a directory specified by the path. If the path is not specified, the default is the directory where the job is submitted

BasisOrder: The order of finite elements

Curved Surfaces: **on** or **off**
on: The surfaces of the finite elements on the model surface are represented by curved surfaces to better approximate the geometry.
off: The surfaces of the finite elements on the model surface are represented by flat surfaces

Solver Specification

```
LinearSolver: {
   Solver:          CG
   Verify: yes
   SaveDOFs: yes
   Preconditioner:    DIAGONAL
   AbsoluteTolerance: 1.0e-18
   Tolerance:        1.0e-10
   MaxIterations:     100000
 }

PicardSolver: {
   Solver:          CG
   Preconditioner:    DIAGONAL
   AbsoluteTolerance: 1.0e-18
   Tolerance:        1.0e-10
   MaxIterations:     100000
 }

NonlinearSolver: {
   Solver:          Newton
   PicardIteration: 10
   MaxIterations:   40
   AbsTolerance:  2.33e-10
   RelTolerance:  1.0e-8
 }

EigenSolver : {
  NumEigenvalues: 5
  FrequencyShift:  1.0
}

HarmonicAnalysis : {
          Start : 50
          End: 250
          Interval: 50
//        Frequency: 50
          StiffnessDamping: 0.01
          MassDamping: 100000
}
```

LinearSolver specifies the parameters for the linear solver.

PicardSolver specifies the parameters for the Picard solver. It is used in conjunction with the Newton solver in **NonlinearSolver**.

For **LinearSolver** and **PicardSolver**, they have the same input parameters.

- **Solver**: Type of solver - **CG**, **GMRES**, **MUMPS**
- **Verify**: yes – calculates the solution residual, no – default option
- **SaveDOFs**: yes – saves degrees of freedom for further post-processing (default), no – don't save
- **Preconditioner**: Type of preconditioner – **DIAGONAL**, **CHOLESKY**, **SSOR**, **ILU**
- **AbsoluteTolerance**: Absolute convergence tolerance
- **Tolerance**: Relative convergence tolerance
- **MaxIterations**: Maximum number of iterations when the solver process terminates even the tolerance has not been reached

NonlinearSolver specifies the parameters for the nonlinear solver

- **Solver**: Type of solver - **Newton**
- **MaxIterations**: Maximum number of iterations when the solver process terminates even the tolerance has not been reached
- **PicardIteration**: Number of iterations for Picard update before switching to Newton update. Using Picard update in the beginning of the nonlinear update stabilizes the nonlinear solution finding process. Once the current solution is sufficiently close to the real solution, using Newton update accelerates the convergence process.
- **AbsTolerance**: Absolute convergence tolerance
- **RelTolerance**: Relative convergence tolerance

EigenSolver specifies how many eigenmodes with frequency above a shifted frequency will be searched. Only homogenous boundary conditions could be used in this case.

- **NumEignvalues**: The number of eigenmodes searched
- **FrequencyShift**: The frequency above which the eigenmodes are calculated [unit in Hz]

HarmonicAnalysis specifies the frequency range for the harmonic response calculations. The **LinearSolver** container to solve the linear system must also be present in this case. The external loadings, specified in the Neumann boundary conditions, are treated as magnitudes at the given frequencies. If the stiffness- or mass-proportional damping coefficients are specified, the problem becomes complex according to the Rayleigh model and appropriate linear solvers must be utilized.

- **Frequency**: The frequency [unit in Hz] at which the harmonic response is calculated. If not provided, the following frequency scan is performed.
- **Start**: Start frequency [unit in Hz]
- **End**: End frequency [unit in Hz]
- **Interval**: Frequency sweep interval [unit in Hz]
- **StiffnessDamping**: stiffness-proportional damping coefficient
- **MassDamping**: mass-proportional damping coefficient

Material Property Specification

```
ThermalConductivity: {
  Id: 1  //VolumeID (copper)
  Value: 391
 }

VolumeMaterial: {
  Id: 1 //VolumeID (copper)
  ElasticLambda: 8.1244e+10      // Either Lambda and mu
  ElasticMu:  4.3363e+1010
  // PoissonsRatio: 0.36          // or Poisson's Ratio/Young's Modulus
  // YoungsModulus: 17e6
  ElasticAlpha : 1.70000e-5           // Not relevant for eigenmode solver
  //Density: 8960                  // Relevant only for eigenmode solver
 }
```

ThermalConductivity specifies the thermal properties of the material.

- **Id:** The volume ID, the block number defined in CUBIT for a certain region of the mesh
- **Value**: Thermal conductivity [unit in W/(m.K)]

VolumeMaterial specifies the structural properties of the material.

- **ElasticLambda**: Lame constant [unit in N/m^2 or Pa]
- **ElasticMu**: Shear modulus [unit in N/m^2 or Pa]
- **YoungsModulus:** Young's modulus [unit in Pa]
- **PoissonsRatio:** Poisson's ratio
- **ElasticAlpha**: Thermal expansion coefficient [unit in 1/K]
- **Density**: Material density [unit in kg/m^3]

In the code λ and μ are used to calculate the Stress-Strain relationship instead of Young's modulus and Poisson's ratio, but in the project file you can specify any of these two. Relationship between (λ, μ) and (E, σ) is based on the following formulation:

$$\mu = \frac{E}{2(1+\sigma)}$$

$$\lambda = \frac{E\sigma}{(1+\sigma)*(1-2\sigma)}$$

where E and σ are Young's modulus and Poisson's ratio, respectively.

Mechanical Analysis Boundary Specification

```
Boundary: {
   Id: X //
   ConditionType: Dirichlet
   DirichletValue: xx
 }

Boundary: {
   Id: 4   //out wall
   ConditionType: Neumann
   NeumannValue: 0.
 }

Boundary: {
   Id: 1   // beam pipe end – fix z direction
   ConditionType: Mixed
   MixedType: NEUMANN NEUMANN DIRICHLET
   MixedValue: 0., 0., 0.
 }

Boundary: {
   Id: 6  // RF Gun Inner surfaces
   ConditionType: LFDetuning
   WhichMode: 0
   Directory: omega3p_results
   Omega3PId: 6  //Exterior surfaces in omega3p_results

   Method: Gradient
   TargetGradient: 60e6   // 60MV/m
   SymmetryFactor: 2
   StartPoint: 0.0001, 0.0001, -0.017018
   EndPoint:   0.0001, 0.0001,  0.017018
}
```

Boundary specifies the boundary conditions in **ElasticProblem**

- **Id:** The surface boundary ID number, defined as the sideset of a surface of the mesh in CUBIT
- **ConditionType:** The type of boundary condition.
  - **Dirichlet**: Value is fixed on the boundary
  - **Neumann**: Normal loading is fixed on the boundary
  - **Mixed**: Combination of Dirichlet and Neumann boundary conditions
  - **LFDetuning**: For simulation of Lorentz force detuning
- **DirichletValue**: Dirichlet value if **ConditionType** is **Dirichlet**; fixed displacement [unit in m]

- **NeumannValue**: Neumann value if **ConditionType** is **Neumann**; normal loading on the surface is fixed. If **NeumannValue** is positive, pressure points into the surface; if **NeumannValue** is negative, pressure points out of the surface [unit in Pa].
- **MixedType**: Boundary condition type in x, y, z directions if **ConditionType** is **Mixed**. Mixed type refers to a combination of Dirichlet and Neumann types. Homogenous **NEUMANN** in this context refers to a free degree of freedom whereas **DIRICHLET** a fixed degree of freedom. In the above example (**NEUMANN**, **DIRICHLET**, **NEUMANN**), in y direction the value is fixed and set to zero displacement while x and z are free to move.
- **MixedValue**: Value of each type in the corresponding direction when **ConditionType** is **Mixed**.

Options must be specified for the **ConditionType: LFDetuning**

- **Omega3PId:** Sideset id (set in Omega3P startup file) of the interface surface between the vacuum region used by the electromagnetic solver Omega3P and the material region used by the mechanical or thermal solver
- **Directory**: Omega3P results directory
- **WhichMode**: Mode id (zero-based numbering) in the Omega3P results, default value is 0
- **Method**: method to be used for the field normalization
    - **Gradient**: if choosen the following options required
        - **TargetGradient** [unit in V/m]
        - **GradientDirection** x, y, z [unit in m] – orthonormal vector, default is 0, 0, 1
        - **SymmetryFactor**: 0 – full cell, 1 – half cell, 2 – quarter cell
        - **StartPoint** x0, y0, z0 [unit in m]
        - **EndPoint** x1, y1, z1 [unit in m]
    - **Voltage**: if choosen the following options required
        - **TargetVoltage** [unit in V]
        - **VoltageDirection** x, y, z [unit in m] – orthonormal vector, default is 0, 0, 1
        - **SymmetryFactor**: 0 – full cell, 1 –half cell, 2 – quarter cell
        - **StartPoint** x0, y0, z0 [unit in m]
        - **EndPoint** x1, y1, z1 [unit in m]
    - **Powerinput:** if choosen the following option required
        - **TargetPowerinput** [unit in w]

Thermal Analysis Boundary Specification

```
Boundary: {
  Id: 2  //2K (see Srf Cavity FPC example)
  ConditionType: Dirichlet
  DirichletValue: 2
 }

Boundary: {
  Id: 4 // out wall
  ConditionType: Neumann
  NeumannValue: 0.
 }

Boundary: {
  Id: 5  // channel 1, 22 Celcius
  ConditionType: Robin
  RobinConstantFactor: 20000.0
  RobinConstantValue: 440000.0
 }
```

Boundary specifies the boundary conditions for **ThermostaticProblem**.

- **Id:** The surface boundary ID number, defined as the sideset of a surface of the mesh in CUBIT
- **ConditionType:** The type of boundary condition
  - **Dirichlet**: Value is fixed on the boundary
  - **Neumann**: Normal derivative is fixed on the boundary
  - **Robin**: Convective boundary condition
  - **RFHeating**: Determined by electromagnetic fields
  - **RFHeatingMap**: Determined by surface power map in input file
- **DirichletValue**: Dirichlet value when **ConditionType** is **Dirichle**t; fixed temperature [unit in K]
- **NeumannValue**: Neumann value when **ConditionType** is **Neumann**; heat flux into the surface [unit in W/m^2]
- **RobinConstantFactor**: Convective coefficient when **ConditionType** is **Robin** [unit in $W/(m^2K)$]
- **RobinConstantValue**: The product of convective coefficient [unit in $W/(m^2K)$]] and ambient temperature [unit in K] when **ConditionType** is **Robin**

For **RFHeating/RFHeatingMap** boundary condition, there are several ways to define the RF heating parameters. Only one needs to use one of the following input methods.

1) **Direct Input Scaling** – RF heating is scaled by the value in **TargetPowerinput**. This method is used if the electromagnetic solver is **S3P**.

```
Boundary:
 {
   Id: 8  // cu_antenna_rf  (see Srf Cavity FPC example)

   ConditionType: RFHeating
   WhichMode: 0
   Directory: s3p_results
   Method: Powerinput
   TargetPowerinput: 437.5
   SurfaceResistance: functions/SRinnerCU
 }
```

2) **Target Gradient Scaling** – The gradient of the electric field is scaled to match the value in **TargetGradient**. The voltage is calculated from the start point to the end point and then divided by the length between the two points.

```
Boundary: {
   Id: 6  // RF GUN Inner surface
   ConditionType: RFHeating  // RFinterface
   WhichMode: 0
   Sigma: 5.8e7
   Directory: omega3p_results
   Method: Gradient
   TargetGradient: 60e6
   DutyFactor: 0.00036
   StartPoint: 0.0001, 0.0001 -0.017018
   EndPoint:   0.0001, 0.0001  0.017018
 }
```

3) **Target Power Loss Scaling** – The surface power loss is calculated using the normalized RF field from **Omega3P**, and then is scaled to match the value in **TargetPowerloss**.

```
Boundary: {
   Id: 6
   ConditionType: RFHeating
   WhichMode: 0
   Directory: omega3p_results
   Method: PowerLoss
   TargetPowerloss: 500
   Sigma: 5.8e7
 }
```

4) Heating Map: One can also specify RF heating by a surface heat flux map supplied by an input file

```
Boundary: {
  Id: 6
  ConditionType: RFHeatingMap
  InputFile: power.txt
 }
```

Boundary: specifies the boundary conditions

- **Id:** The surface boundary ID number, defined as the sideset of a surface of the mesh in CUBIT
- **ConditionType:** Type of boundary condition is **RFHeating/RFHeatingMap**
- **Method:** For normalizing the fields obtained by **Omega3P** or **S3P**
  - **Powerinput**: Input power into the system [unit in W]
  - **Gradient**: Value between two points along a line [unit in V/m]
  - **PowerLoss**: Total power loss on metallic surfaces [unit in W]
- **WhichMode**: Mode ID number obtained from the electromagnetic solver; starting from 0
- **Directory**: The result directory of the electromagnetic solver
- **TargetPowerloss**: Total power loss on metallic surface when **Method** is **PowerLoss** [unit in W]
- **TargetPowerinput**: Total input power to the system when **Method** is **Powerinput** [unit in W]
- **TargetGradient:** Accelerating gradient when **Method** is **Gradient** [unit in V/m]
- **StartPoint**: (x, y, z) coordinates of start point for gradient integration when **Method** is **Gradient** [units in m]
- **EndPoint**: (x, y, z) coordinates of end point for gradient integration when **Method** is **Gradient** [units in m]
- **Sigma**: Electric conductivity [unit in S/m]
- **DutyFactor**: Fraction of time during operation for average power calculation
- **SymmetryFactor**: **1** for full cell and **0** for half cell
- **InputFile**: Name of the file for surface heat flux map with format (*ix, iy, theta, phi, F*) specified in spherical coordinate angles (theta, phi) for map vertices. division MxN is the division grid, *ix* is the grid ID number in the theta direction (starting from 1 to M), *iy* the grid ID in the phi direction (starting from 1 to N), *theta* the corresponding theta coordinate, *phi* the corresponding phi coordinate, and *F* the heat flux at the grid point [units in W/m^2], respectively. The data file is filled each row by advancing *ix* from 1 to M for a fixed *iy* (starting from 1), and then repeat for the next *iy* until *iy* = N.

Lossy Dielectric Heat

```
HeatSource: {
```

```
  Id: 5   //window volume
  ConditionType: RFHeating  // RFinterface
  WhichMode: 0
  DielectricConstantE: 9
  LossTangentE: 0.0001
  Directory: s3p_results
  Method: Powerinput
  TargetPowerinput: 437.5
}
```

HeatSource defines the heating due to dielectric loss or external volume heating.

- **Id:** The dielectric volume ID, the block ID defined in CUBIT
- **ConditionType:** Type of boundary condition is **RFHeating**.
- **WhichMode**: Mode ID number obtained from the electromagnetic solver; starting from 0
- **DielectricConstantE**: The dielectric constant.
- **LossTangent**: The dielectric loss tangent.
- **Directory**: The result directory of the electromagnetic solver
- **Method:** Use **Powerinput**, Total input power to the system when **Method** is **Powerinput** [unit in W]
- **TargetPowerinput**: Total input power to the system when **Method** is **Powerinput** [unit in W]

External Volume Heating

```
HeatSource: {
   Id: 9                                  // block ID of external heating volume
   ConditionType: ExtVHeating             // external volume heating
   ExtVHeatingFile:  heating_file.dat     // file of the external heating map
}
```

HeatSource defines the heating due to dielectric loss or external volume heating.

- **Id:** The external heating volume ID, the volume block ID defined in CUBIT
- **ConditionType:** Type of heating is **ExtVHeating**.
- **ExtVHeatingFile**: File name of the external volume heating data

External volume heating data file format (heating_file.dat) :

{start of file}
Dummy line (FLUKA output)
Dummy line
Dummy line
Dummy line
Dummy line
Dummy line
r1 r2 nr dr
theta1 theta2 ntheta dtheta
z1 z2 nz dz
power_scaling_factor
dummy line
P1 P2 P3 P4 P5
P6 P7 ….
…
P_(nr*ntheta*nz)
{end of file}

The volume heating handles the FLUKA output only.

- **nr, ntheta, nz**: data intervals in r, theta and z respectively
- **r1, r2:** range of data in radial direction (unit in m)
- **theta1, theta2:** range of data in azimuthal direction (unit in radian)
- **z1, z2:** range of data in z direction (unit in m)
- **power_scaling_factor:** scaling factor to the data (to the desired power value)
- **P1, P2, …:** power density at the middle of the grid cells, total nr*ntheta*nz points. The data is in the following order:
  loop in "r" first
  loop in "theta" second
  loop in "z" last

Thermal Shell Specifications

```
Shell: {
  Bd:  6              //  where thermal shell id is
  Material: 221    // Material property id for shell, copper
  BasisOrder: 1
  Thickness: 1e-5 // Thickness of the shell
}

ThermalConductivity: {
   Id: 221 // shell, copper
   Function: functions/CURRR100
}
```

A shell element with high thermal conductivity is used to represent a realistic copper coating configuration. It acts as a thermal conduction path for otherwise poor thermal conductors. It is used to calculate the solution of nonlinear thermal steady state. A small thickness is given for a shell element.

Shell defines the use of shell elements with the following parameters.

- **Bd:** The surface boundary ID number, defined as the sideset of a surface of the mesh in CUBIT
- **Material**: Material property id, see **ThermalConductivity** below.
- **BasisOrder**: Order of basis functions of shell elements
- Thickness**:** Thickness of the shell [unit in m]

ThermalConductivity specifies the thermal conductivity shell elements.

- **Id:** Material property id
- **Function**: File that defines the nonlinear thermal conductivity of shell elements

Nonlinear Material Property Specification

Three types of nonlinear material are available for thermal analysis.

Nonlinear thermal conductivity **-** Variation of thermal conductivity over large temperature differential can be significant. Especially for superconductivity, thermal conductivity could vary by a few orders of magnitude over a few degrees.

```
ThermalConductivity: {
   Id: 3 // SS  VolumeID
   Function: functions/ss304
 }
```

Nonlinear surface resistance **-** Surface resistance contributes to heat generation and is strongly dependent on the nonlinear profile of thermal conductivity.

```
 Boundary: {
    Id: 8 //cu_rf
    ConditionType: RFHeating
    WhichMode: 0
    Directory: s3p_results
    Method: Powerinput
    TargetPowerinput: 437.5
    SurfaceResistance: functions/SRInnerCU
}
```

Nonlinear convective surface **-** Proper representation of convective surface is important for both linear and nonlinear thermal analysis.

```
Boundary: {
    Id:  x  //SurfaceID
    ConditionType: Robin
    RobinFactor: functions/He2p2f
    RobinValue: functions/He2p2v
}
```

ThermalConductivity specifies the thermal conductivity of the material.

- **Id:** The volume ID, the block number defined in CUBIT for a certain region of the mesh
- **Function:** File that defines the function for the material non-linearity

Boundary specifies the boundary condition

- **Id:** The surface boundary ID number, defined as the sideset of a surface of the mesh in CUBIT
- **ConditionType:** Type of boundary condition
  - **Robin**: Convective boundary condition
  - **RFHeating**: Determined by electromagnetic fields
- **WhichMode**: Mode ID number obtained from the electromagnetic solver; starting from 0
- **Directory**: The result directory of the electromagnetic solver
- **Method:** Use **Powerinput**, Total input power to the system when **Method** is **Powerinput** [unit in W]
- **SurfaceResistance**: File that defines the non-linear surface resistance
- **RobinFactor:** File that defines the nonlinear Robin factor, convective coefficient [unit in $W/(m^2K)$]
- **RobinValue**: File that defines the nonlinear Robin value, the product of convective coefficient [unit in $W/(m^2K)$]] and ambient temperature [unit in K]

Nonlinear function definitions

Nonlinear function specification adheres to the following format.

$$T_{start} \quad T_{end} \quad f(T)$$

where $T_{start} < T < T_{end}$, $K(T) = f(T)$. In the following example, between 2.8322 and 2.9743 K, thermal conductivity of Niobium is defined as

$$K(T) = 3.0832 * T - 5.8522$$

Data format

-1000.0 2.832165577 2.879956447
*2.832165577 2.974324853 3.083208397*T+(-5.852200243)*
*2.974324853 3.113087159 3.639414105*T+(-7.506536704)*
*3.113087159 3.515595987 4.036830127*T+(-8.743727419)*
*3.515595987 3.774013419 5.115074289*T+(-12.53439827)*
*3.774013419 4.065133383 6.332495619*T+(-17.1289627)*
*4.065133383 4.445771322 7.412071590*T+(-21.51758302)*
*4.445771322 4.903279388 10.121085200*T+(-33.56123806)*
*4.903279388 5.380539698 9.369006979*T+(-29.8735884)*
*5.380539698 5.924230122 10.741888110*T+(-37.26042984)*
*5.924230122 6.511852578 8.822751514*T+(-25.89102299)*
*6.511852578 7.133625729 5.027705228*T+(-1.17824105)*
*7.133625729 7.814767829 6.925084079*T+(-14.71343164)*
*7.814767829 8.589912105 9.991724201*T+(-38.67851221)*
*8.589912105 9.522008394 -4.121996501*T+(82.55710809)*

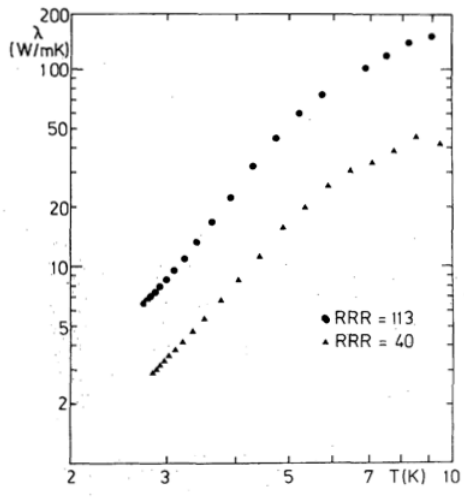

Fig. 1: Thermal conductivity of reactor grade niobium (RRR=40) and niobium of higher purity (RRR=113)

Note: The above data is tabulated from Fig. 1 in the paper, “SUPERCONDUCTING NIOBIUM CAVITIES OF IMPROVED THERMAL CONDUCTIVITY” by H.Lenge1er and W.Weingarten, G. Miiller and H.Piel.

Mesh Deformation Specification

TEM3P calculates structural deformation caused by structural displacement or radiation pressure arising from electromagnetic fields. It interpolates structural deformation to determine the corresponding deformation on the inner vacuum mesh. The frequency shift can be obtained by running **Omega3P** on the deformed mesh.

```
MeshDump: {
  WriteDeformedMesh: on
  WriteDeformedEMMesh: on
  MeshDeformScale: 1.0
  EMMeshInputDir: omega3p_results
  WriteStressStrain: on
}
```

MeshDump specifies the output of the deformed mesh and stress/strain .mod files.

- **WriteDeformedMesh: On** or **Off**. If **On**, the deformed mesh is written.
- **WriteDeformedEMMesh: On** or **Off**. If **On**, the deformed vacuum mesh is written.
- **MeshDeformScale:** Deformation scale factor, 1.0 is a default value. This option is useful to write out the deformed mesh for an arbitrary phase for the eigenmode and harmonic response analysis.
- **EMMeshInputDir:** Directory the deformed mesh is written.

Time-Domain Thermal Analysis Specification

```
ThermostaticProblem: {

… …

 // Time domain

 Transient:
 {
  TimeStepping:
  {
   Type: Thermal
   Scheme: BackwardEuler
   MaximumTime: 8.e1
   DT:        4.e-1
   InitialT:    0.
   InitialTemp: 25.
  }

  Excitation:
  {
   Power: 1.
   Pulse:
   {
    Type: Monochromatic
        Frequency: 1.
        Rise periods: 10
        T0: 0.
   }
  }
 }

 … …
}
```

Tranisent specifies time domain module in **ThermostaticProblem**, which is incidentally used for its code infrastructure.

- **TimeStepping** specifies time advancement scheme.
  - **Type**: Type of solver
    - **Thermal**: Thermal solver
  - **Scheme**: Time advancement scheme
    - **BackwardEuler**: Backward Euler scheme
  - **MaximumTime**: Maximum runtime [unit in s]
  - **DT**: Time step [unit in s]
  - **InitialT**: Start time, default 0 [unit in s]

- **InitialTemp**: Initial temperature, default 0 [unit in ^{o}C]

- **Excitation** specifies properties of excitation.
  - **Power**: Power of pulse [unit in W]
  - **Pulse**: Type of excitation
    - **Monochromatic**: Single-frequency pulse
      - **Frequency**: Drive frequency [unit in Hz]
      - **Rise periods**: Number of cycles to reach steady state from initial zero value
      - **Fall periods**: Number of cycles to zero value from flattop (optional)
      - **T0**: Start time of pulse [unit in s]
      - **TMax**: Total time of pulse [unit in s] (optional)