ElmerSolver Command File

Thomas Zwinger

`thomas.zwinger[at]csc.fi`

Computational Environment & Application

CSC–Scientific Computing Ltd.

The Finnish IT center for science

Espoo, Finland

CSC

Contents

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) . . .

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) . . .

. . . but simply also composed using a text editor

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) ...

... but simply also composed using a text editor

The Rules:

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) ...

... but simply also composed using a text editor

The Rules:

- comments start with `!`

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) . . .

. . . but simply also composed using a text editor

The Rules:

- comments start with `!`
- Important: do not use tabulators for indents!

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) . . .

. . . but simply also composed using a text editor

The Rules:

- comments start with `!`
- Important: do not use tabulators for indents!
- a section always ends with the keyword `End`

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) . . .

. . . but simply also composed using a text editor

The Rules:

- comments start with `!`
- Important: do not use tabulators for indents!
- a section always ends with the keyword `End`
- parameters (except from Elmer keyword database) need to be casted by their types: `Integer` `Real` `Logical` `String` `File`

CSC

The Solver Input File (SIF)

- contains all the information for the solution step, `ElmerSolver_mpi`
- can be exported by `ElmerGUI` (also `ElmerFront`) ...

... but simply also composed using a text editor

The Rules:

- comments start with `!`
- Important: do not use tabulators for indents!
- a section always ends with the keyword `End`
- parameters (except from Elmer keyword database) need to be casted by their types: `Integer` `Real` `Logical` `String` `File`
- `Parametername(n,m)` indicates a $n \times m$ array

Header

The header declares where to search for the mesh database

```
Header
   Mesh DB "." "dirname"
End
```

preceding path + directory name of mesh database

Constants

Declaration of constant values that can be obtained from within **every** solver and boundary condition **subroutine** or **function**, can be declared.

```
Constants
   Gas Constant = Real 8.314E00
   Gravity (4) = 0 -1 0 9.81
End
```

a scalar constant

Gravity vector, an array with a registered name

Simulation

Principle declarations for simulation

```
Simulation
   Coordinate System = "Cartesian 2D"
   Coordinate Mapping(3) = Integer 1 2 3
   Simulation Type ="Steady"
   Output Intervals = 1
   Steady State Max Iterations = 10
   Steady State Min Iterations = 2
   Output File = "name.result"
   Post File = "name.ep"
   max output level = n
End
```

- Coordinate System — choices: `Cartesian {1D,2D,3D}`, `Polar {2D,3D}`, `Cylindric`, `Cylindric Symmetric`, `Axi Symmetric`
- Coordinate Mapping — permute, if you want to interchange directions
- Simulation Type — either `Steady` or `Transient`
- Output Intervals — how often you want to have results
- Steady State Max Iterations — maximum rounds on one time level
- Steady State Min Iterations — minimum rounds on one Timestep
- Output File — contains data to restart run
- Post File — `ElmerPost`-file
- max output level — n=1 talkative like a Finnish lumberjack, n=42 all and everything

Solver

Example: (Navier) Stokes solver

```
Solver 1
   Equation = "Navier-Stokes"
   Linear System Solver = "Direct"
   Linear System Direct Method = "UMFPack"
   Linear System Convergence Tolerance = 1.0E-06
   Linear System Abort Not Converged = True
   Steady State Convergence Tolerance = 1.0E-03
   Stabilization Method = Stabilized
   Nonlinear System Convergence Tolerance = 1.0E-05
   Nonlinear System Max Iterations = 40
   Nonlinear System Min Iterations = 1
   Nonlinear System Newton After Iterations = 30
   Nonlinear System Newton After Tolerance = 1.0E-05
End
```

name of the solver

alt. `Iterative`

not used

a non-linear problem

Newton iter.

Body

Here the different bodies (there can be more than one) get their `Equation`, `Material`, `Body Force` and `Initial Condition` assigned

```
Body 2
   Name = "identifier"
   Equation = 1
   Material = 2
   Body Force = 1
   Initial Condition = 1
End
```

there can be more than one body

give the body a name

one `Equation/Material/`

`Body Force/Initial Condition`

can serve several bodies

Equation

- set active solvers
- give keywords for the behaviour of different solvers

```
Equation 1
  Active Solvers(2) = 1 2
  Convection = Computed
  Flow Solution Name = String "Flow Solution"
  NS Convect = False
End
```

Bodyforce

- declares the solver-specific $\mathbf{f}$ from $\mathbf{A} \cdot \mathbf{\Psi} = \mathbf{f}$ for the body
- body force can also be a dependent function (see later).

Here for the (Navier) Stokes solver

```
Body Force 1
   Flow BodyForce 1 = 0.0
   Flow BodyForce 2 = -9.81 !  good old gravity
End
```

CSC

Material

- sets material properties for the body.
- material properties can be scalars or tensors and also
- can be given as dependent function/expression

```
Material 1
   Density = 918.0
   Heat Capacity = Variable Temperature
    MATC "2.1275D03 + 7.253D00*(tx - 273.16)"
   My Variable = Real 1002.0
End
```

dependence

a MATC expression (see later)

not in keyword DB!

Initial Conditions

- initializes variable values
- sets initial guess for steady state simulation
- sets initial value for transient simulation
- variable values can be functions/expressions

```
Initial Condition 1
   Velocity 1 = 0.0
   Velocity 2 = Variable Coordinate 1
    MATC "initialvelocity(tx)"
   Pressure = 0.0
   My Variable = Real 0.0
End
```

dependence

a MATC function (see later)

not in keyword DB

Boundary Conditions

- Dirichlet: `variablename` = *value*
- Neumann: often enabled with keyword (e.g., HTEqu. `Heat Flux BC = True`) followed by the flux value
- No BC $\equiv$ Natural BC!
- values can be given as functions

Example: (Navier) Stokes with no penetration (normal) and free slip (tangential)

```
Boundary Condition 1
 Name = "slip"
 Target Boundaries = 4
 Normal-Tangential Velocity = Logical True
 Velocity 1 = Real 0.0
End
```

name

refers to boundary no. 4 in mesh

components with respect to surface normal

normal component

Bodies on Boundaries

- need to solve (dimension-1) PDEs (e.g., kinematic BC on free surface)

CSC

Bodies on Boundaries

- need to solve (dimension-1) PDEs (e.g., kinematic BC on free surface)
- need to define the (dimension-1) entity as a separate body

CSC

Bodies on Boundaries

- need to solve (dimension-1) PDEs (e.g., kinematic BC on free surface)
- need to define the (dimension-1) entity as a separate body
- in the corresponding `Boundary`-section:
  `Body ID = n` with $n >$ highest occurring body in the mesh

Bodies on Boundaries

- need to solve (dimension-1) PDEs (e.g., kinematic BC on free surface)
- need to define the (dimension-1) entity as a separate body
- in the corresponding `Boundary`-section:
  `Body ID = n` with $n >$ highest occurring body in the mesh
- define `Body Force`, `Material`, `Equation` and `Initial Condition` for that body

Bodies on Boundaries

- need to solve (dimension-1) PDEs (e.g., kinematic BC on free surface)
- need to define the (dimension-1) entity as a separate body
- in the corresponding `Boundary`-section:
  `Body ID = n` with $n >$ highest occurring body in the mesh
- define `Body Force`, `Material`, `Equation` and `Initial Condition` for that body
- full dimensional metric is still valid on the BC body $\Rightarrow$ has to be taken into account in user supplied subroutines

Tables and Arrays

- **Tables** may be used for piecewise linear dependency of a variable

Tables and Arrays

- **Tables** may be used for piecewise linear dependency of a variable

```
Density   = Variable Temperature
    Real
       0   900
     273   1000
     300   1020
     400   1000
     End
```

Tables and Arrays

- **Tables** may be used for piecewise linear dependency of a variable

```
Density   = Variable Temperature
   Real
      0   900
    273   1000
    300   1020
    400   1000
    End
```

- **Arrays** may be used to declare vector/tensor parameters

Tables and Arrays

- **Tables** may be used for piecewise linear dependency of a variable

```
Density   = Variable Temperature
   Real
      0   900
    273   1000
    300   1020
    400   1000
    End
```

- **Arrays** may be used to declare vector/tensor parameters

```
Target Boundaries(3) = 2 4 5
My Parameter Array(3,3) = Real 1 2 3 \
                               4 5 6 \
                                 7 8 9
```

MATC

- library for the numerical evaluation of mathematical expressions

MATC

- library for the numerical evaluation of mathematical expressions
- defined in SIF for use in `ElmerSolver`

MATC

- library for the numerical evaluation of mathematical expressions
- defined in SIF for use in `ElmerSolver`

or by `ElmerPost` as post-processing feature

e.g. K $\longrightarrow$ $^\circ$C: `math Celsius = Temperature + 273.16`

MATC

- library for the numerical evaluation of mathematical expressions
- defined in SIF for use in `ElmerSolver`

or by `ElmerPost` as post-processing feature
e.g. K $\longrightarrow$ $^\circ$C: `math Celsius = Temperature + 273.16`

- very close to C-syntax

CSC

MATC

- library for the numerical evaluation of mathematical expressions
- defined in SIF for use in `ElmerSolver`

or by `ElmerPost` as post-processing feature
e.g. K $\longrightarrow$ $^{\circ}$C: `math Celsius = Temperature + 273.16`

- very close to C-syntax

also logical evaluations (if) and loops (for)

MATC

- library for the numerical evaluation of mathematical expressions
- defined in SIF for use in `ElmerSolver`

or by `ElmerPost` as post-processing feature

e.g. K $\longrightarrow$ $^{\circ}$C: `math Celsius = Temperature + 273.16`

- very close to C-syntax

  also logical evaluations (if) and loops (for)
- documentation on Funet (MATC Manual)

CSC

MATC contd.

- simple numerical evaluation:

```
Viscosity Exponent = Real MATC "1.0/3.0"
```
or
```
Viscosity Exponent = Real $1.0/3.0
```

MATC contd.

- simple numerical evaluation:

```
Viscosity Exponent = Real MATC "1.0/3.0"
```
or
```
Viscosity Exponent = Real $1.0/3.0
```

- as an expression dependent on a variable:

```
Heat Capacity = Variable Temperature
Real MATC "2.1275D03 + 7.253D00*(tx - 273.16)"
```

CSC

MATC contd.

- simple numerical evaluation:

```
Viscosity Exponent = Real MATC "1.0/3.0"
```

or

```
Viscosity Exponent = Real $1.0/3.0
```

- as an expression dependent on a variable:

```
Heat Capacity = Variable Temperature
Real MATC "2.1275D03 + 7.253D00*(tx - 273.16)"
```

- as an expression of multiple variables:

```
Temp = Variable Latitude, Coordinate 3
Real MATC "49.13 + 273.16 - 0.7576 * tx(0) - 7.992E-03 * tx(1)"
```

CSC

MATC contd.

- simple numerical evaluation:

```
Viscosity Exponent = Real MATC "1.0/3.0"
```
or
```
Viscosity Exponent = Real $1.0/3.0
```

- as an expression dependent on a variable:

```
Heat Capacity = Variable Temperature
Real MATC "2.1275D03 + 7.253D00*(tx - 273.16)"
```

- as an expression of multiple variables:

```
Temp = Variable Latitude, Coordinate 3
Real MATC "49.13 + 273.16 - 0.7576 * tx(0) - 7.992E-03 * tx(1)"
```

- as function defined at the top of SIF:

```
$ function stemp(X) { _stemp = 49.13 + 273.16 - 0.7576*X(0)
                                            - 7.992E-03*X(1) }
```

```
Temp = Variable Latitude, Coordinate 3
Real MATC "stemp(tx)"
```

User Defi ned Functions

Example: $\rho(T(^\circ C)) = 1000 \cdot [1 - 10^{-4} \cdot (T - 273.0)]$

User Defined Functions

Example: $\rho(T(^\circ C)) = 1000 \cdot [1 - 10^{-4} \cdot (T - 273.0)]$

```
FUNCTION getdensity( Model, n, T ) RESULT(dens)
USE DefUtils
IMPLICIT None
 TYPE(Model_t) ::  Model

 INTEGER ::  n

 REAL(KIND=dp) ::  T, dens

 dens = 1000*(1-1.0d-4(T-273.0d0))

END FUNCTION getdensity
```

CSC

User Defi ned Functions

Example: $\rho(T(^\circ C)) = 1000 \cdot [1 - 10^{-4} \cdot (T - 273.0)]$

```
FUNCTION getdensity( Model, n, T ) RESULT(dens)
USE DefUtils
IMPLICIT None
 TYPE(Model_t) ::  Model

 INTEGER ::  n

 REAL(KIND=dp) ::  T, dens

 dens = 1000*(1-1.0d-4(T-273.0d0))

END FUNCTION getdensity
```

compile: `elmerf90 mydensity.f90 -o mydensity`

User Defined Functions

Example: $\rho(T(^\circ C)) = 1000 \cdot [1 - 10^{-4} \cdot (T - 273.0)]$

```
FUNCTION getdensity( Model, n, T ) RESULT(dens)
USE DefUtils
IMPLICIT None
 TYPE(Model_t) ::  Model

 INTEGER ::  n

 REAL(KIND=dp) ::  T, dens

 dens = 1000*(1-1.0d-4(T-273.0d0))

END FUNCTION getdensity
```

compile: `elmerf90 mydensity.f90 -o mydensity`

in SIF:

```
Density = Variable Temperature
Procedure "mydensity" "getdensity"
```

User Defined Subroutines

```
RECURSIVE SUBROUTINE &
mysolver( Model,Solver,dt,TransientSimulation )
TYPE(Model_t) :: Model
TYPE(Solver_t) :: Solver
REAL(KIND=dp) :: dt
LOGICAL :: TransientSimulation
...
assembly, solution
...
END SUBROUTINE mysolver
```

User Defined Subroutines

```
RECURSIVE SUBROUTINE &
mysolver( Model,Solver,dt,TransientSimulation )
TYPE(Model_t) ::  Model
TYPE(Solver_t) ::  Solver
REAL(KIND=dp) ::  dt
LOGICAL ::  TransientSimulation
...
assembly, solution
...
END SUBROUTINE mysolver
```

`Model`	pointer to the whole Model (solvers, variables)
`Solver`	pointer to the particular solver
`dt`	current time step size
`TransientSimulation`	`.TRUE.` if transient simulation

User Defined Subroutines

```
RECURSIVE SUBROUTINE &
mysolver( Model,Solver,dt,TransientSimulation )
TYPE(Model_t) :: Model
TYPE(Solver_t) :: Solver
REAL(KIND=dp) :: dt
LOGICAL :: TransientSimulation
...
assembly, solution
...
END SUBROUTINE mysolver
```

`Model`	pointer to the whole Model (solvers, variables)
`Solver`	pointer to the particular solver
`dt`	current time step size
`TransientSimulation`	`.TRUE.` if transient simulation

compile:

```
elmerf90 mysolverfile.f90 -o mysolverexe
```

User Defined Subroutines

```
RECURSIVE SUBROUTINE &
mysolver( Model,Solver,dt,TransientSimulation )
TYPE(Model_t) :: Model
TYPE(Solver_t) :: Solver
REAL(KIND=dp) :: dt
LOGICAL :: TransientSimulation
...
assembly, solution
...
END SUBROUTINE mysolver
```

`Model`	pointer to the whole Model (solvers, variables)
`Solver`	pointer to the particular solver
`dt`	current time step size
`TransientSimulation`	`.TRUE.` if transient simulation

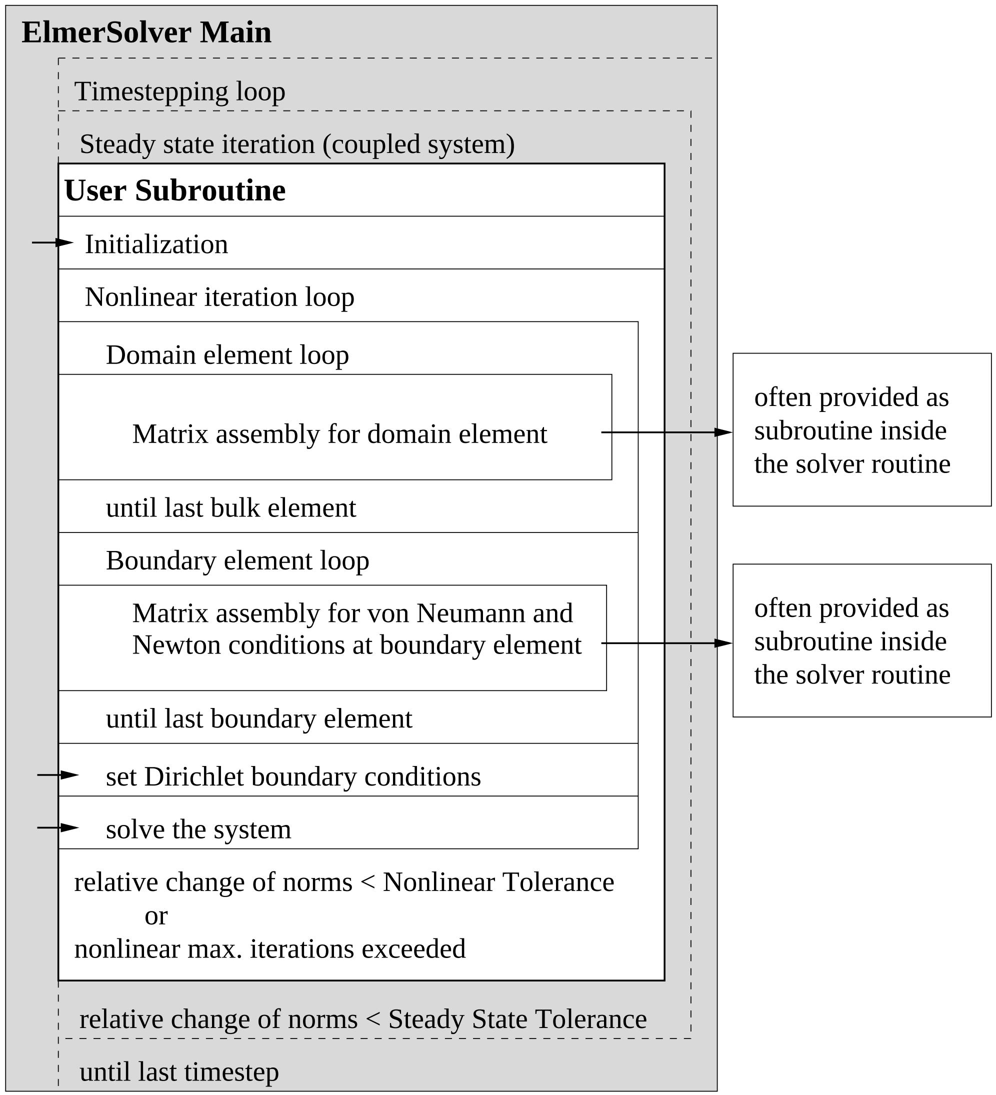
compile:

```
elmerf90 mysolverfile.f90 -o mysolverexe
```

SIF:

```
Procedure = "/path/to/mysolverexe" "mysolver"
```

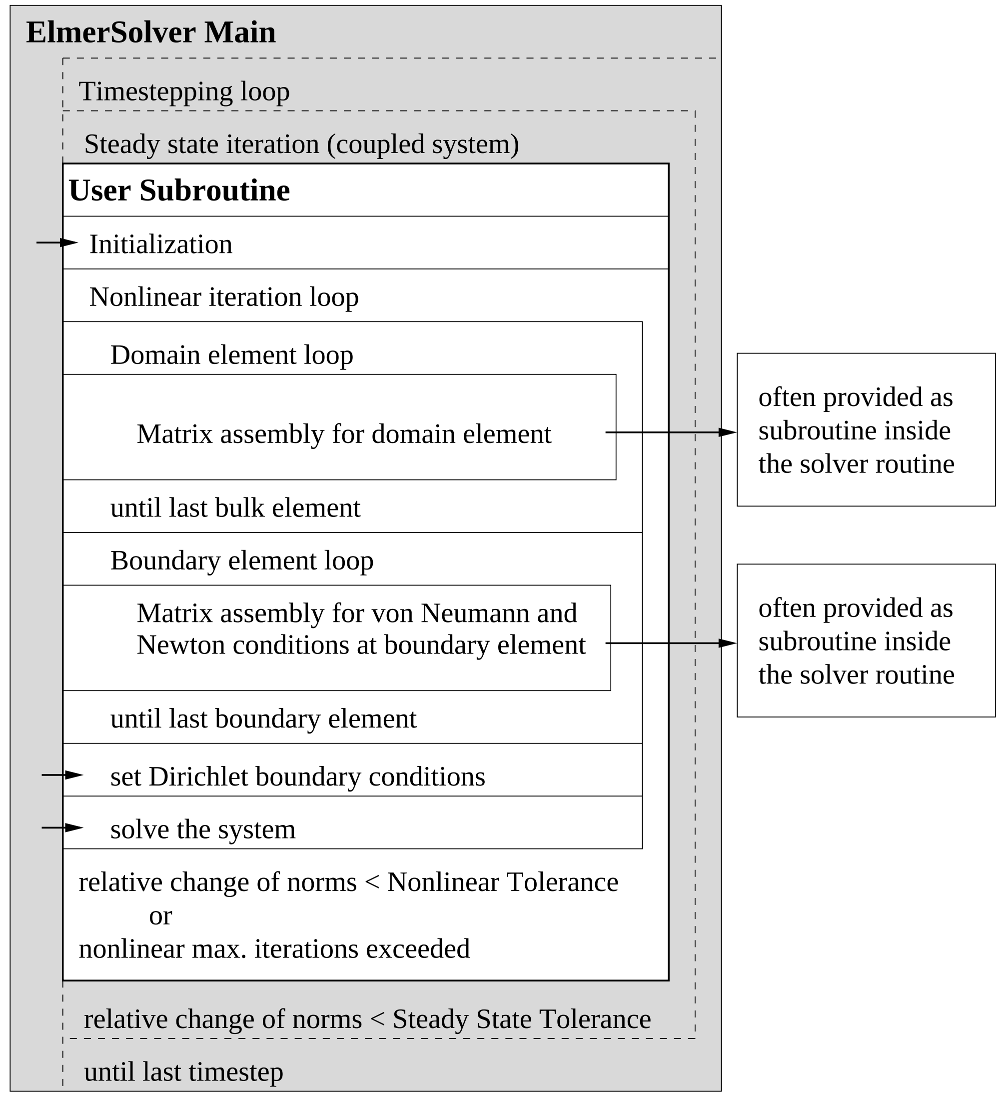
User Defi ned Subroutines contd.



User Defi ned Subroutines contd.

Pre-defined routines

- `CALL DefaultInitialize()`
- `CALL DefaultUpdateEquations( STIFF, FORCE )`
- `CALL DefaultFinishAssembly()`
- `CALL DefaultDirichletBCs()`
- `Norm = DefaultSolve()`



Multiple Meshes

- In the `Header`, declare the *global* mesh database

```
Mesh DB "." "dirname"
```

Multiple Meshes

- In the `Header`, declare the *global* mesh database

```
Mesh DB "." "dirname"
```

- In the `Solver`, declare the *local* mesh the solver is run on:

```
Mesh = File "/path/to/" "mesh"
```

CSC

Multiple Meshes

- In the `Header`, declare the *global* mesh database

```
Mesh DB "." "dirname"
```

- In the `Solver`, declare the *local* mesh the solver is run on:

```
Mesh = File "/path/to/" "mesh"
```

- variable values will be interpolated

Multiple Meshes

- In the `Header`, declare the *global* mesh database

```
Mesh DB "." "dirname"
```

- In the `Solver`, declare the *local* mesh the solver is run on:

```
Mesh = File "/path/to/" "mesh"
```

- variable values will be interpolated

⚠ they will boldly be extrapolated, should your meshes not be congruent!

Element Types

- In section `Equation`:

Element Types

- In section `Equation`:

```
Element = [n:#dofs d:#dofs p:#dofs b:#dofs e:#dofs f:#dofs]
```

`n` . . . nodal, `d` . . . DG element, `p` `p-element`, `b` . . . bubble, `e` . . . edge, `f` . . . face DOFs

Element Types

- In section `Equation`:

```
Element = [n:#dofs d:#dofs p:#dofs b:#dofs e:#dofs f:#dofs]
```

`n` . . . nodal, `d` . . . DG element, `p` `p-element`, `b` . . . bubble, `e` . . . edge, `f` . . . face DOFs

- `Element = [d:0]` . . . DG DOFs ≡ mesh element nodes

Element Types

- In section `Equation`:

```
Element = [n:#dofs d:#dofs p:#dofs b:#dofs e:#dofs f:#dofs]
```

`n` . . . nodal, `d` . . . DG element, `p` `p-element`, `b` . . . bubble, `e` . . . edge, `f` . . . face DOFs

- `Element = [d:0]` . . . DG DOFs ≡ mesh element nodes
- If `Equation` applies to more than one solver, `Element = ...` applies for all solver.

CSC

Element Types

- In section `Equation`:

```
Element = [n:#dofs d:#dofs p:#dofs b:#dofs e:#dofs f:#dofs]
```

n . . . nodal, d . . . DG element, p `p-element`, b . . . bubble, e . . . edge, f . . . face DOFs

- `Element = [d:0]` . . . DG DOFs ≡ mesh element nodes
- If `Equation` applies to more than one solver, `Element = ...` applies for all solver.

  selectively for each solver:

```
Element[1] = ...
Element[2] = ...
⋮
Element[n] = ...
```

Specialities

- given names for components of vector fields:

```
Variable = var_name[cname 1:#dofs cname 2:#dofs ...  ]
```

Specialities

- given names for components of vector fields:

```
Variable = var_name[cname 1:#dofs cname 2:#dofs ... ]
```

- ”internal” Solver can be run as external Procedure (enabling definition of variable names)

```
Procedure = "FlowSolve" "FlowSolver"
```

```
Variable = Flow[Veloc:3 Pres:1]
```

Specialities

- given names for components of vector fields:

```
Variable = var_name[cname 1:#dofs cname 2:#dofs ...  ]
```

- ”internal” Solver can be run as external Procedure (enabling definition of variable names)

```
Procedure = "FlowSolve" "FlowSolver"
```

```
Variable = Flow[Veloc:3 Pres:1]
```

- Residuals of solver variables (e.g., Navier Stokes):

```
Procedure = "FlowSolve" "FlowSolver"
```

```
Variable = Flow[Veloc:3 Pres:1]
```

```
Exported Variable 1 = Flow Loads[Stress Vector:3 CEQ Residual:1]
```

Specialities

- given names for components of vector fields:

```
Variable = var_name[cname 1:#dofs cname 2:#dofs ...  ]
```

- ”internal” Solver can be run as external Procedure (enabling definition of variable names)

```
Procedure = "FlowSolve" "FlowSolver"
```

```
Variable = Flow[Veloc:3 Pres:1]
```

- Residuals of solver variables (e.g., Navier Stokes):

```
Procedure = "FlowSolve" "FlowSolver"
```

```
Variable = Flow[Veloc:3 Pres:1]
```

```
Exported Variable 1 = Flow Loads[Stress Vector:3 CEQ Residual:1]
```

- Solver execution:

```
Exec Solver = {Before Simulation, After Simulation, Never, Always}
```

Elmer Parallel Version

- Pre-processing: `ElmerGrid` with options:

Partition by direction:

`-partition 2 2 1 0`	First partition elements (default)
`-partition 2 2 1 1`	First partition nodes

$2 \times 2 \times = 4$

Partition using METIS:

`-metis` n `0`	PartMeshNodal (default)
`-metis` n `1`	PartGraphRecursive
`-metis` n `2`	PartGraphKway
`-metis` n `3`	PartGraphVKway

Elmer Parallel Version

- Pre-processing: `ElmerGrid` with options:

  Partition by direction:

`-partition 2 2 1 0`	First partition elements (default)
`-partition 2 2 1 1`	First partition nodes
$2 \times 2 \times = 4$	

  Partition using METIS:

`-metis` *n* `0`	PartMeshNodal (default)
`-metis` *n* `1`	PartGraphRecursive
`-metis` *n* `2`	PartGraphKway
`-metis` *n* `3`	PartGraphVKway

- Execution: `mpirun -np` *n* `ElmerSolver_mpi`

Elmer Parallel Version

- Pre-processing: `ElmerGrid` with options:

  Partition by direction:

`-partition 2 2 1 0`	First partition elements (default)
`-partition 2 2 1 1`	First partition nodes
$2 \times 2 \times = 4$	

  Partition using METIS:

`-metis` *n* `0`	PartMeshNodal (default)
`-metis` *n* `1`	PartGraphRecursive
`-metis` *n* `2`	PartGraphKway
`-metis` *n* `3`	PartGraphVKway

- Execution: `mpirun -np n ElmerSolver_mpi`

- Combining parallel results: in mesh-database directory

  `ElmerGrid 15 3 name`