Elmer
Software Development Practices
APIs for Solver and UDF

ElmerTeam
CSC – IT Center for Science

CSC, November.2015
Elmer programming languages

- Fortran90 (and newer)
  - ElmerSolver (~240,000 lines of which ~50% in DLLs)
- C++
  - ElmerGUI (~18,000 lines)
  - ElmerSolver (~10,000 lines)
- C
  - ElmerPost (~45,000 lines)
  - ElmerGrid (~30,000 lines)
  - MATC (~11,000 lines)
Tools for Elmer development

- Programming languages
  - Fortran90 (and newer), C, C++
- Compilation
  - Compiler (e.g. gnu), configure, automake, make, (cmake)
- Editing
  - emacs, vi, notepad++,...
- Code hosting (git)
  - Current: https://github.com/ElmerCSC
  - Obsolete: www.sf.net/projects/elmerfem
- Consistency tests
- Code documentation
  - Doxygen
- Theory documentation
  - Latex
- Community server
  - www.elmerfem.org (forum, wiki, etc.)
Elmer libraries

**ElmerSolver**
- Required: Matc, HutIter, Lapack, Blas, Umfpack (GPL)
- Optional: Arpack, Mumps, Hypre, Pardiso, Trilinos, SuperLU, Cholmod, NetCDF, HDF5, ...

**ElmerGUI**
- Required: Qt, ElmerGrid, Netgen
- Optional: Tetgen, OpenCASCADE, VTK, QVT
Elmer licenses

ElmerSolver library is published under LGPL
  - Enables linking with all license types
  - It is possible to make a new solver even under proprietary license
  - Note: some optional libraries may constrain this freedom due to use of GPL licences

Rest of Elmer is published under GPL
  - Derived work must also be under same license (“copyleft”)
Elmer version control at GitHub

In 2015 the official version control of Elmer was transferred from svn at sf.net to git hosted at GitHub.

Git offers more flexibility over svn:
- Distributed version control system
- Easier to maintain several development branches
- More options and hence also steeper learning curve
- Developed by Linus Torvalds to host Linux kernel development

GitHub is a portal providing Git and some additional services:
- Management of user rights
- Controlling pull requests
Elmer uses git version control system for the code repository and development

- Hosted at github
- Development version in "trunk" is considered stable
- To obtain the whole source code

  ```
git clone https://github.com/ElmerCSC/elmerfem.git
  ```

- Git client available in command line in *nix systems
- In Windows systems a nice graphical client is "Tortoise"
Elmer at Github

https://github.com/ElmerCSC
Activity on Github

Git clones

<table>
<thead>
<tr>
<th>Date</th>
<th>Clones</th>
<th>Unique Cloners</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Visitors

<table>
<thead>
<tr>
<th>Date</th>
<th>Views</th>
<th>Unique Visitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Elmer is published under (L)GPL

- Used worldwide by thousands of researchers (?)
- One of the most popular open source multiphysical software
<table>
<thead>
<tr>
<th>Country / Territory</th>
<th>Visits</th>
<th>Pages / Visit</th>
<th>Avg. Visit Duration</th>
<th>% New Visits</th>
<th>Bounce Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>9,611</td>
<td>3.52</td>
<td>00:03:31</td>
<td>55.59%</td>
<td>60.55%</td>
</tr>
<tr>
<td>Germany</td>
<td>8,938</td>
<td>5.08</td>
<td>00:05:52</td>
<td>31.76%</td>
<td>51.92%</td>
</tr>
<tr>
<td>France</td>
<td>4,404</td>
<td>3.15</td>
<td>00:03:06</td>
<td>30.68%</td>
<td>68.66%</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>4,028</td>
<td>4.19</td>
<td>00:04:27</td>
<td>39.03%</td>
<td>53.87%</td>
</tr>
<tr>
<td>Finland</td>
<td>3,841</td>
<td>7.06</td>
<td>00:09:14</td>
<td>20.57%</td>
<td>22.49%</td>
</tr>
<tr>
<td>Italy</td>
<td>3,602</td>
<td>4.44</td>
<td>00:03:38</td>
<td>55.19%</td>
<td>47.47%</td>
</tr>
<tr>
<td>Spain</td>
<td>1,957</td>
<td>6.16</td>
<td>00:05:52</td>
<td>36.54%</td>
<td>46.65%</td>
</tr>
<tr>
<td>Japan</td>
<td>1,484</td>
<td>3.14</td>
<td>00:03:20</td>
<td>36.59%</td>
<td>64.08%</td>
</tr>
<tr>
<td>India</td>
<td>1,341</td>
<td>2.86</td>
<td>00:02:39</td>
<td>73.38%</td>
<td>69.57%</td>
</tr>
<tr>
<td>Canada</td>
<td>1,335</td>
<td>3.12</td>
<td>00:02:53</td>
<td>58.13%</td>
<td>63.00%</td>
</tr>
<tr>
<td>Netherlands</td>
<td>1,333</td>
<td>3.73</td>
<td>00:03:44</td>
<td>37.21%</td>
<td>58.21%</td>
</tr>
<tr>
<td>Austria</td>
<td>1,120</td>
<td>4.88</td>
<td>00:04:43</td>
<td>29.29%</td>
<td>53.04%</td>
</tr>
<tr>
<td>Switzerland</td>
<td>995</td>
<td>3.63</td>
<td>00:03:23</td>
<td>40.80%</td>
<td>59.30%</td>
</tr>
<tr>
<td>China</td>
<td>975</td>
<td>4.14</td>
<td>00:04:04</td>
<td>48.92%</td>
<td>53.54%</td>
</tr>
<tr>
<td>Poland</td>
<td>907</td>
<td>4.06</td>
<td>00:03:59</td>
<td>41.23%</td>
<td>55.35%</td>
</tr>
<tr>
<td>Brazil</td>
<td>876</td>
<td>2.98</td>
<td>00:03:16</td>
<td>42.01%</td>
<td>64.50%</td>
</tr>
<tr>
<td>Russia</td>
<td>831</td>
<td>2.97</td>
<td>00:02:53</td>
<td>51.50%</td>
<td>65.82%</td>
</tr>
</tbody>
</table>
~20k Windows downloads at sf.net in a year

![Map showing global downloads](image)

### Downloads

**19,185**

In the selected date range

### Top Country

**United States**

16% of下载者

### Top OS

**Windows**

93% of下载者

### OS Downloads as:

<table>
<thead>
<tr>
<th>Country</th>
<th>Android</th>
<th>BSD</th>
<th>Linux</th>
<th>Macintosh</th>
<th>Unknown</th>
<th>Windows</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>3%</td>
<td>1%</td>
<td>80%</td>
<td>3,182</td>
</tr>
<tr>
<td>Germany</td>
<td>0%</td>
<td>0%</td>
<td>4%</td>
<td>1%</td>
<td>0%</td>
<td>80%</td>
<td>2,313</td>
</tr>
<tr>
<td>Italy</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>1%</td>
<td>0%</td>
<td>80%</td>
<td>1,537</td>
</tr>
<tr>
<td>France</td>
<td>0%</td>
<td>0%</td>
<td>4%</td>
<td>1%</td>
<td>1%</td>
<td>79%</td>
<td>798</td>
</tr>
<tr>
<td>India</td>
<td>0%</td>
<td>0%</td>
<td>6%</td>
<td>1%</td>
<td>4%</td>
<td>78%</td>
<td>782</td>
</tr>
<tr>
<td>Russia</td>
<td>0%</td>
<td>0%</td>
<td>4%</td>
<td>0%</td>
<td>0%</td>
<td>77%</td>
<td>772</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>2%</td>
<td>0%</td>
<td>81%</td>
<td>642</td>
</tr>
<tr>
<td>China</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>1%</td>
<td>1%</td>
<td>78%</td>
<td>637</td>
</tr>
<tr>
<td>Japan</td>
<td>0%</td>
<td>0%</td>
<td>2%</td>
<td>2%</td>
<td>0%</td>
<td>77%</td>
<td>699</td>
</tr>
<tr>
<td>Spain</td>
<td>0%</td>
<td>0%</td>
<td>5%</td>
<td>0%</td>
<td>20%</td>
<td>63%</td>
<td>561</td>
</tr>
<tr>
<td>Poland</td>
<td>0%</td>
<td>0%</td>
<td>2%</td>
<td>0%</td>
<td>0%</td>
<td>87%</td>
<td>532</td>
</tr>
<tr>
<td>Canada</td>
<td>1%</td>
<td>0%</td>
<td>2%</td>
<td>2%</td>
<td>0%</td>
<td>85%</td>
<td>410</td>
</tr>
<tr>
<td>Brazil</td>
<td>0%</td>
<td>0%</td>
<td>4%</td>
<td>1%</td>
<td>0%</td>
<td>88%</td>
<td>391</td>
</tr>
<tr>
<td>Finland</td>
<td>0%</td>
<td>0%</td>
<td>2%</td>
<td>1%</td>
<td>0%</td>
<td>78%</td>
<td>300</td>
</tr>
</tbody>
</table>
16k Windows downloads at sf.net in a year

<table>
<thead>
<tr>
<th>Country</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>2,553</td>
</tr>
<tr>
<td>Germany</td>
<td>2,529</td>
</tr>
<tr>
<td>Italy</td>
<td>1,342</td>
</tr>
<tr>
<td>Russia</td>
<td>975</td>
</tr>
<tr>
<td>Japan</td>
<td>789</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>609</td>
</tr>
<tr>
<td>France</td>
<td>548</td>
</tr>
<tr>
<td>China</td>
<td>529</td>
</tr>
<tr>
<td>India</td>
<td>483</td>
</tr>
<tr>
<td>Spain</td>
<td>400</td>
</tr>
<tr>
<td>Poland</td>
<td>385</td>
</tr>
<tr>
<td>Finland</td>
<td>305</td>
</tr>
</tbody>
</table>

Date Range: 2011-06-01 to 2012-06-01
Installers

- Fresh Windows installers
  - Currently only 64 bit version
  - Also a parallel version with msmpi

- Elmer for Debian & Ubuntu etc. at launchpad
  - Nightly builds from Git repository
  - To install
    
    ```
    $ sudo apt-add-repository ppa:juhmat/elmer-test
    $ sudo apt-get update
    $ sudo apt-get install elmerfem-csc
    ```
Cmake build system

During 2014-2015 Elmer was migrated from gnu autotools to **cmake**

cmake offers several advantages

- Enables cross compilation for different platforms (e.g. Intel MICs)
- More standardizes installation scripts
- Straight-forward package creation for many systems (using cpack)
- Great testing utility with ctest

Transition to cmake required significant code changes

- ISO C-bindings & many changes in APIs
- Backward compatibility in compilation lost
Obtaining the source code

To clone the code (this is anonymously):

```bash
  git clone https://github.com/ElmerCSC/elmerfem.git
```

– We work with branches. To change into another branch:
  ```bash
  cd elmerfem
  git checkout branchname
  ```

We use the following branches (confined to most important):

– `release`: contains stable release (~half-yearly update)
– `devel`: our main branch from which you should get latest updates
– `elmerice`: the main developer branch for Elmer/Ice

NB: you might have to do a

```bash
  git checkout --track origin/elmerice
```
Obtaining the source code

On branches:
# Code organization

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fem</td>
<td>Source of ElmerSolver</td>
</tr>
<tr>
<td>matc</td>
<td>MATC language</td>
</tr>
<tr>
<td>fhutiter</td>
<td>Fortran version of linear algebra solvers</td>
</tr>
<tr>
<td>ElmerGUI</td>
<td>Graphical User Interface to Elmer based on QT4</td>
</tr>
<tr>
<td>elmerice</td>
<td>Elmer/Ice solver and function source code</td>
</tr>
<tr>
<td>post</td>
<td>Legacy visualization tool</td>
</tr>
<tr>
<td>elmergrid</td>
<td>Grid manipulation for Elmer</td>
</tr>
<tr>
<td>mathlibs</td>
<td>Contains Lapack/BLAS from netlib (avoid using them)</td>
</tr>
<tr>
<td>elmerparam</td>
<td>Additional package for optimization</td>
</tr>
</tbody>
</table>
Compilation of the whole code with cmake

To compile the whole code see example scripts under [www.csc.fi/elmer](http://www.csc.fi/elmer) and [www.elmerfem.org](http://www.elmerfem.org)

```
ELMERSRC="/path/to/sourcecode/elmerfem"
BUILDDIR="/path/too/existing/and/empty/builddir"
IDIR="/path/to/installation/dir/"
TOOLCHAIN="/path/to/optional/toolchainfile.smake"
cmake $ELMERSRC \
  -DCMAKE_BUILD_TYPE=DEBUG \n  -DCMAKE_INSTALL_PREFIX=$IDIR \n  -DWITH_MPI:BOOL=TRUE \n  -DWITH_Mumps:BOOL=TRUE \n  -DWITH_Hypre:BOOL=TRUE \n  -DWITH_ELMERGUI:BOOL=TRUE \n  -DWITH_OCC:BOOL=TRUE \n  -DWITH_PARAVIEW:BOOL=TRUE \n  -DWITH_PYTHONQT:BOOL=TRUE \n  -DWITH_QWT:BOOL=TRUE \n  -DWITH_VTK:BOOL=TRUE \n  -DWITH_PYTHONQT:BOOL=FALSE \n  -DWITH_MATC:BOOL=TRUE \n  -DWITH_ElmerIce:BOOL=TRUE
```
Consistency tests

- Simple shell script to run through the cases + piece of C-code to compare the norm of solutions
- There are >300 consistency tests (November 2015)
  - Located under fem/tests, run with ctest in build-directory
- Each time a significant commit is made the tests are run with the fresh version
  - Aim: trunk version is a stable version
  - New tests for each major new feature
- The consistency tests provide a good starting point for taking some Solver into use
  - cut-paste from sif file
- Note: the consistency tests have often poor time and space resolution for rapid execution
Consistency tests - example

```
zwinger@elmeruser-VM64bit ~/Source/Elmer_devel/elmerfem $ cd ..
zwinger@elmeruser-VM64bit ~/Source/Elmer_devel $ cd builddir_elmerice/
zwinger@elmeruser-VM64bit ~/Source/Elmer_devel/builddir_elmerice $ ctest -LE elmerice

Test project /home/zwinger/Source/Elmer_devel/builddir_elmerice
  Start 35: mgdyn_lamstack_widefreq_harmonic
  1/379 Test #35: mgdyn_lamstack_widefreq_harmonic .... Passed 12.58 sec
  Start 36: NaturalConvectionRestart
  2/379 Test #36: NaturalConvectionRestart ............ Passed 10.90 sec
  Start 37: ContactPatch3D
  3/379 Test #37: ContactPatch3D ..................... Passed 4.28 sec
  Start 38: ContactPatch3D_np4
  4/379 Test #38: ContactPatch3D_np4 .................. Passed 6.13 sec
  Start 39: CurvedBndryPFEM
  5/379 Test #39: CurvedBndryPFEM ..................... Passed 0.12 sec
  Start 40: heateq
  6/379 Test #40: heateq ............................ Passed 0.42 sec
  Start 41: StrainCalculation02
  7/379 Test #41: StrainCalculation02 ................ Passed 8.62 sec
  Start 42: ContactPatch2Dtoo
  8/379 Test #42: ContactPatch2Dtoo .................. Passed 0.16 sec
  Start 43: freesurf
  9/379 Test #43: freesurf ........................... Passed 1.61 sec
  Start 44: rotflow
```
Elmer finite element software

Here is a list of all modules:

- Elmer library
  - Default API
- Dynamically linked solvers
- Dynamically linked functions
- Utility programs
  - Program ResultToPost
  - Program ResultToResult
  - Program ViewFactors
Special comment indicators: !> and <! 

Doxygen – Example in code

!> Subroutine for computing fluxes and gradients of scalar fields.
!> For example, one may compute the the heat flux as the negative gradient of temperature
!> field multiplied by the heat conductivity.
!> \ingroup Solvers

SUBROUTINE FluxSolver( Model,Solver,dt,Transient )

USE CoordinateSystems
USE DefUtils
IMPLICIT NONE

TYPE(Solver_t) :: Solver  !< Linear & nonlinear equation solver options
TYPE(Model_t) :: Model    !< All model information (mesh, materials, BCs, etc...)
REAL(KIND=dp) :: dt       !< Timestep size for time dependent simulations
LOGICAL :: Transient      !< Steady state or transient simulation

!    Local variables

TYPE(ValueList_t),POINTER :: SolverParams
subroutine FluxSolver ( TYPE(Model_t) Model,
                     TYPE(Solver_t) Solver,
                     REAL(KIND=dp) dt,
                     LOGICAL Transient )

Subroutine for computing fluxes and gradients of scalar fields. For example, one may compute the heat flux as the negative gradient of temperature field multiplied by the heat conductivity.

**Parameters:**
- **Solver**: Linear & nonlinear equation solver options
- **Model**: All model information (mesh, materials, BCs, etc...)
- **dt**: Timestep size for time dependent simulations
- **Transient**: Steady state or transient simulation

References **BulkAssembly**().

Here is the call graph for this function:

![Call Graph](FluxSolver -> BulkAssembly)
Compilation of a DLL module

- Applies both to Solvers and User Defined Functions (UDF)
- Assumes that there is a working compile environment that provides "elmerf90" script
  - Comes with the Windows installer, and Linux packages
  - Generated automatically when ElmerSolver is compiled

elmerf90 MySolver.f90 -o MySolver.so
User defined function API

!------------------------------------------------------

!> Standard API for UDF

!------------------------------------------------------

RECURSIVE FUNCTION MyProperty( Model, n, t ) RESULT(f)

!------------------------------------------------------

USE DefUtils
IMPLICIT NONE

!------------------------------------------------------

TYPE(Model_t) :: Model !< Handle to all data
INTEGER :: n !< Current node
REAL(KIND=dp) :: t !< Parameter(s)
REAL(KIND=dp) :: f !< Parameter value at node

Actual code ...
Function API

MyProperty = Variable time
"MyModule" "MyProperty"

User defined function (UDF) typically returns a real valued property at a given point

It can be located in any section that is used to fetch these values from a list
  – Boundary Condition, Initial Condition, Material,...
Solver API

!------------------------------------------------------
!> Standard API for Solver
!------------------------------------------------------
SUBROUTINE MySolver( Model,Solver,dt,Transient )
!------------------------------------------------------

USE DefUtils
IMPLICIT NONE

!------------------------------------------------------
TYPE(Solver_t) :: Solver !< Current solver
TYPE(Model_t) :: Model !< Handle to all data
REAL(KIND=dp) :: dt !< Timestep size
LOGICAL :: Transient !< Time-dependent or not
!------------------------------------------------------

Actual code ...
Solver API

Solver 1

  Equation = "MySolver"
  Procedure = "MyModule" "MySolver"
  ...

End

Solver is typically a FEM implementation of a physical equation (PDE)

But it could also be an auxiliary solver that does something completely different

Solver is usually called once for each coupled system iteration
Elmer – High level abstractions

The quite good success of Elmer as a multi-physics code may be addressed to certain design choices

– Solver is an abstract dynamically loaded object
– Parameter value is an abstract property fetched from a list

The abstractions mean that new solvers may be implemented without much need to touch the main library

– Minimizes need of central planning
– Several applications fields may live their life quite independently (electromagnetics and glaciology)

MATC – a poor man’s Matlab adds to flexibility as algebraic expressions may be evaluated on-the-fly
Solver as an abstract object

Solver is a dynamically loaded object (.dll or .so)
  – May be developed and compiled separately
Solver utilizes heavily common library utilities
  – Most common ones have interfaces in DefUtils
Any solver has a handle to all of the data
Typically a solver solves a weak form of a differential equation
Currently ~50 different Solvers, roughly half presenting physical phenomena
  – No upper limit to the number of Solvers
Solvers may be active in different domains, and even meshes
The menu structure of each solver in ElmerGUI may be defined by an .xml file
Properties as abstract objects

Properties are saved in a list structure by their name.

Namespace of properties is not fixed; they may be introduced in the command file:

- E.g. "MyProperty = Real 1.23" adds a property "MyProperty" to a list structure related to the solver block.

In the code, parameters are fetched from the list:

- E.g. "val = GetReal( Material,'MyProperty',Found)" retrieves the above value 1.23 from the list.

A "Real" property may be any of the following:

- Constant value
- Linear or cubic dependence via table of values
- Expression given by MATC (MatLab/C-type command language)
- User defined functions with arbitrary dependencies
- Real vector or tensor

As a result, solvers may be weakly coupled without any a priori defined manner.

There is a price to pay for the generic approach but usually it is less than 10%.

*SOLVER.KEYWORDS* file may be used to give the types for the keywords in the command file.
DefUtils

DefUtils module includes wrappers to the basic tasks common to standard solvers

- E.g. "DefaultDirichlet()" sets Dirichlet boundary conditions to the given variable of the Solver
- E.g. "DefaultSolve()" solves linear systems with all available direct, iterative and multilevel solvers, both in serial and parallel

Programming new Solvers and UDFs may usually be done without knowledge of other modules
# DefUtils – some functions

## Public Member Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE(Solver_t) function, pointer</td>
<td><code>GetSolver()</code></td>
<td>(Solver) Returns the selected solver.</td>
</tr>
<tr>
<td>TYPE(Matrix_t) function, pointer</td>
<td><code>GetMatrix()</code> (USolver)</td>
<td>(Matrix) Returns the selected matrix.</td>
</tr>
<tr>
<td>TYPE(Mesh_t) function, pointer</td>
<td><code>GetMesh()</code> (USolver)</td>
<td>(Mesh) Returns the selected mesh.</td>
</tr>
<tr>
<td>TYPE(Element_t) function, pointer</td>
<td><code>GetCurrentElement()</code> (Element)</td>
<td>(Element) Returns the current element.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetElementIndex()</code> (Element)</td>
<td>(Element Index) Returns the index of the selected element.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetNOFActive()</code> (USolver)</td>
<td>(NumberOfActive) Returns the number of active elements.</td>
</tr>
<tr>
<td>REAL(KIND=dp) function</td>
<td><code>GetTime()</code></td>
<td>(Time) Returns the current time.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetTimeStep()</code></td>
<td>(TimeStep) Returns the current time step.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetTimeStepInterval()</code></td>
<td>(TimeStepInterval) Returns the time step interval.</td>
</tr>
<tr>
<td>REAL(KIND=dp) function</td>
<td><code>GetTimestepSize()</code></td>
<td>(TimestepSize) Returns the size of the current time step.</td>
</tr>
<tr>
<td>REAL(KIND=dp) function</td>
<td><code>GetAngularFrequency()</code> (ValueList, Found)</td>
<td>(AngularFrequency) Returns the angular frequency.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetCoupledIter()</code></td>
<td>(CoupledIter) Returns the current coupled iteration.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetNonlinIter()</code></td>
<td>(NonlinIter) Returns the current non-linear iteration.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetNOFBoundaryElements()</code> (UMesh)</td>
<td>(BoundaryElements) Returns the number of boundary elements.</td>
</tr>
<tr>
<td>subroutine</td>
<td><code>GetScalarLocalSolution()</code> (x, name, UElement, USolver, tStep)</td>
<td>(ScalarLocalSolution) Returns the scalar local solution.</td>
</tr>
<tr>
<td>subroutine</td>
<td><code>GetVectorLocalSolution()</code> (x, name, UElement, USolver, tStep)</td>
<td>(VectorLocalSolution) Returns the vector local solution.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetNofEigenModes()</code> (name, USolver)</td>
<td>(EigenModes) Returns the number of eigen modes.</td>
</tr>
<tr>
<td>subroutine</td>
<td><code>GetScalarLocalEigenmode()</code> (x, name, UElement, USolver, NoEigen, ComplexPart)</td>
<td>(ScalarLocalEigenmode) Returns the scalar local eigenmode.</td>
</tr>
<tr>
<td>subroutine</td>
<td><code>GetVectorLocalEigenmode()</code> (x, name, UElement, USolver, NoEigen, ComplexPart)</td>
<td>(VectorLocalEigenmode) Returns the vector local eigenmode.</td>
</tr>
<tr>
<td>CHARACTER(LEN=MAX_NAME_LEN) function</td>
<td><code>GetString()</code> (List, Name, Found)</td>
<td>(GetString) Returns the string associated with the given list, name, and found status.</td>
</tr>
<tr>
<td>INTEGER function</td>
<td><code>GetInteger()</code> (List, Name, Found)</td>
<td>(GetInteger) Returns the integer associated with the given list, name, and found status.</td>
</tr>
<tr>
<td>LOGICAL function</td>
<td><code>GetLogical()</code> (List, Name, Found)</td>
<td>(GetLogical) Returns the logical associated with the given list, name, and found status.</td>
</tr>
<tr>
<td>recursive REAL(KIND=dp) function</td>
<td><code>GetConstReal()</code> (List, Name, Found, x, y, z)</td>
<td>(GetConstReal) Returns the constant real associated with the given list, name, and found status.</td>
</tr>
<tr>
<td>recursive REAL(KIND=dp) function</td>
<td><code>GetCReal()</code> (List, Name, Found)</td>
<td>(GetCReal) Returns the complex real associated with the given list, name, and found status.</td>
</tr>
<tr>
<td>recursive REAL(KIND=dp) function, dimension(:), pointer</td>
<td><code>GetReal()</code> (List, Name, Found, UElement)</td>
<td>(GetReal) Returns the real associated with the given list, name, found status, and element.</td>
</tr>
</tbody>
</table>
Example: Poisson equation

\[ -\nabla^2 \phi = \rho \]

- Implemented as an dynamically linked solver
  - Available under tests/1dtests

- Compilation by:
  `elmerf90 Poisson.f90 -o Poisson.so`

- Execution by:
  `ElmerSolver case.sif`

- The example is ready to go massively parallel and with all a plethora of elementtypes in 1D, 2D and 3D
Poisson equation: code Poisson.f90

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; Solve the Poisson equation -\nabla \cdot \nabla \phi = \rho</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
</tr>
</tbody>
</table>

SUBROUTINE PoissonSolver( Model, Solver, dt, TransientSimulation )

USE DefUtils
IMPLICIT NONE

!Initialize the system and do the assembly:
!------------------------------------------
CALL DefaultInitialize()

active = GetNOFActive()
DO t=1,active
   Element => GetActiveElement(t)
   n = GetElementNOFNodes()
   LOAD = 0.0d0
   BodyForce => GetBodyForce()
   IF ( ASSOCIATED(BodyForce) ) &
      Load(1:n) = GetReal( BodyForce, 'Source', Found )
   ! Get element local matrix and rhs vector:
   !----------------------------------------
   CALL LocalMatrix( STIFF, FORCE, LOAD, Element, n )
   ! Update global matrix and rhs vector from local contribs
   !---------------------------------------------------------------------
   CALL DefaultUpdateEquations( STIFF, FORCE )
END DO

CALL DefaultFinishAssembly()
CALL DefaultDirichletBCs()
Norm = DefaultSolve()

Solver 1
Equation = "Poisson"
Variable = "Potential"
Variable DOFs = 1
Procedure = "Poisson" "PoissonSolver"
Linear System Solver = "Direct"
Linear System Direct Method = umfpack
Steady State Convergence Tolerance = 1e-09
End

Body Force 1
Source = Variable Potential
Real Procedure "Source" "Source"
End

Boundary Condition 1
Target Boundaries(2) = 1 2
Potential = Real 0
End
CONTAINS

!--------------------------------------------------------------
SUBROUTINE LocalMatrix( STIFF, FORCE, LOAD, Element, n )
!--------------------------------------------------------------

... 

CALL GetElementNodes( Nodes )
STIFF = 0.0d0
FORCE = 0.0d0

! Numerical integration:
!----------------------
DO t=1,IP % n
  ! Basis function values & derivatives at the integration point:
  !-----------------------------------------------
  stat = ElementInfo( Element, Nodes, IP % U(t), IP % V(t), &
                   IP % W(t), detJ, Basis, dBasisdx )

  ! The source term at the integration point:
  !-----------------------------------------------
  LoadAtIP = SUM( Basis(1:n) * LOAD(1:n) )

  ! Finally, the elemental matrix & vector:
  !-----------------------------------------------
  STIFF(1:n,1:n) = STIFF(1:n,1:n) + IP % s(t) * DetJ * &
      MATMUL( dBasisdx, TRANSPOSE( dBasisdx ) )
  FORCE(1:n) = FORCE(1:n) + IP % s(t) * LoadAtIP * Basis(1:n)
END DO

END SUBROUTINE LocalMatrix

!--------------------------------------------------------------
END SUBROUTINE PoissonSolver
!--------------------------------------------------------------

\[ -\nabla \cdot \nabla \Phi = \rho \]

\[ -\int_V \nabla \cdot \nabla \Phi \varphi dV = \int_V \rho \varphi dV \]

\[ -\int_V \nabla \cdot (\nabla \Phi \varphi) dV + \int_V \nabla \Phi \cdot \nabla \varphi dV = \int_V \rho \varphi dV \]

\[ -\oint_{\partial V} \left\{\nabla \Phi \cdot n\right\} \varphi dV + \int_V \nabla \Phi \cdot \nabla \varphi dV = \int_V \rho \varphi dV \]
Poisson equation: source term, examples

Constant source:

Source = 1.0

Source depending piecewise linear on x:

Source = Variable Coordinate 1
Real
  0.0  0.0
  1.0  3.0
  2.0  4.0
End

Source depending on x and y:

Source = Variable Coordinate
Real MATC "sin(2*pi*tx(0))*cos(2*pi(tx(1)))"

Source depending on anything

Source = Variable Coordinate 1
Procedure "Source" "MySource"
Poisson equation: ElmerGUI menus

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE edf>
<edf version="1.0">
  <PDE Name="Poisson">
    <Name>Poisson</Name>
    <BodyForce>
      <Parameter Widget="Label">
        <Name>Properties</Name>
      </Parameter>
      <Parameter Widget="Edit">
        <Name>Source</Name>
        <Type>String</Type>
        <Whatis>Give the source term.</Whatis>
      </Parameter>
    </BodyForce>
    <Solver>
      <Parameter Widget="Edit">
        <Name>Procedure</Name>
        <DefaultValue>"Poisson" "PoissonSolver"</DefaultValue>
      </Parameter>
      <Parameter Widget="Edit">
        <Name>Variable</Name>
        <DefaultValue>Potential</DefaultValue>
      </Parameter>
    </Solver>
    <BoundaryCondition>
      <Parameter Widget="Label">
        <Name>Dirichlet conditions</Name>
      </Parameter>
      <Parameter Widget="Edit">
        <Name>Potential</Name>
        <Whatis>Give potential value for this boundary.</Whatis>
      </Parameter>
    </BoundaryCondition>
  </PDE>
</edf>
Development tools for ElmerSolver

- **Basic use**
  - Editor (emacs, vi, notepad++, jEdit, ...)
  - elmerf90 script

- **Advanced**
  - Editor
  - git client
  - Compiler suite (gfortran, ifort, pathf90, pgf90, ...)
  - Documentation tools (Doxygen, LaTeX)
  - Debugger (gdb)
  - Profiling tools
  - ...

Elmer – some best practices

- Use version control when possible
  - If the code is left to your own local disk, you might as well not write it at all
  - Never fork! (user base of 1000’s)

- Always make a consistency test for a new feature
  - Always be backward compatible
  - If not, implement a warning in the code

- Maximize the level of abstraction
  - Essential for multi-physics software
  - E.g. any number of physical equations, any number of computational meshes, any number of physical or numerical parameters – without the need for recompilation