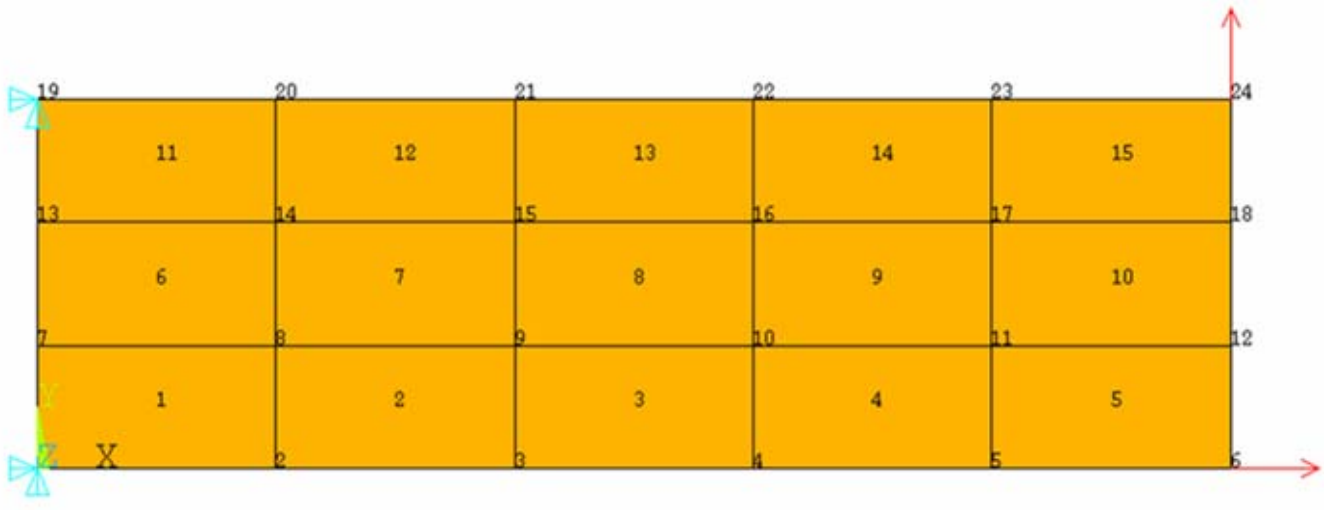
FELyX
Finite Element Library eXperiment

Build your own FE Model completely in FELyX

Generate Nodes, Elements, Material
Introduce loadcase
Solve
Write results into a txt-fiel



Problem

- Build your own, simple FE Model completely in FELyX
  - Create Nodes
  - Connect Nodes to Elements
  - Add an isotropic Material
  - Introduce loadcase
  - Solve
  - Write results into a txt-file
- Three new files are created (felyx/tutorial folder) like in the Motorbike-Frame example
  - MyStructObject.h
  - MyStructObject.cc
  - MyStructObjectApp.cpp
- Create new target, add files to this target and make it active

New StructObject

- A new class MyStructObject is generated in the MyStructObject.h file with the functions
  - void create_model();
  - void print_model();
  - void Write_Solution(std::string, std::string);

```
//
// Header file for MyStructObject
//

#ifndef MyStructObject_h
#define MyStructObject_h MyStructObject_h

#ifdef HAVE_CONFIG_H
#include <config.h>
//#include <fstream.h>                                  Include basics
#endif

// include FELyX object
#include "StructObject.h"                               Include class StructObject

using namespace std;                                    Define namespace
using namespace felyx;

class MyStructObject : public StructObject {           Definition of MyStructObject; Class is derived from StuctObject

  public:

    MyStructObject() : StructObject(2) {}               Empty constructor, initialize StructObject with noise parameter

    void create_model();                                Creat FE-model from scratch

    void print_model();                                 Print model data

    void Write_Solution(std::string, std::string);

};


#endif
```

MyStructObject.cc

- Implement the functions create_model

```
#include "MyStructObject.h"

void MyStructObject::create_model() {

  std::cout << "Create FE model\n";


  double length = 100.0;
  unsigned n_ele_length = 5;

  double width = 30.0;
  unsigned n_ele_width = 3;

  unsigned n_ele = n_ele_length * n_ele_width;
  unsigned n_nodes = ( n_ele_length + 1 ) * ( n_ele_width + 1 );

  Nodes.resize( n_nodes );

  std::vector<node_type>::iterator nit = Nodes.begin();


  for (unsigned i = 0; i < n_ele_width +1; ++i) {
    for (unsigned j = 0; j < n_ele_length +1; ++j ) {
      // set node coordinates
      nit->set(j*length/(n_ele_length),i*width/n_ele_width,0.0);
      // go to next node
      ++nit;
    }
  }

  Materials.push_back( new fe_base::IsotropicMaterial() );
  Materials[0] -> Set("E",70000.);
  Materials[0] -> Set("nu",0.3);

  Elements.resize(n_ele);
  PtrVector<element_type*>::iterator elemit = Elements.begin();
  (*elemit) = new fe_base::Plane182;
  unsigned n0;

  for(unsigned j = 0; j < n_ele_width; ++j) {

    for(unsigned i = 0; i < n_ele_length; ++i) {
      (*elemit) = new fe_base::Plane182;

      n0 = (n_ele_length + 1)*j + i;

      (*elemit)->SetNodeIter(0,Nodes.begin() + n0);
      (*elemit)->SetNodeIter(1,Nodes.begin() + n0 + 1);
      (*elemit)->SetNodeIter(2,Nodes.begin() + n0 + n_ele_length + 2);
      (*elemit)->SetNodeIter(3,Nodes.begin() + n0 + n_ele_length + 1);

      ( *elemit ) -> SetMaterialPtr( Materials[ 0 ] );
      ++elemit;
    }
  }
```

Implementation file for MyStructObject

Implementation of function to create FE model

Data of rectangle to create

Create nodes of model
Resize node container

We need a node iterator

Fill in coordinates of nodes

Create material
Set material parameters

Create elements

Reassign points to the elements

```
  BoundaryConditions.resize(3);
  BoundaryConditions[0].set(Dx,0.);
  BoundaryConditions[0].set(Dy,0.);

  Nodes[0].set(&(BoundaryConditions[0]));

Nodes[(n_ele_length+1)*n_ele_width].set(&BoundaryConditions[0]);

  BoundaryConditions[1].set(Fx,100.);
  BoundaryConditions[2].set(Fy,100.);
  Nodes[n_ele_length].set(&BoundaryConditions[1]);
  Nodes[n_nodes-1].set(&BoundaryConditions[2]);

}
```

Create and assign boundary conditions

Define boundary condition set (Fix support)

Assign boundary condition to a node

Define second boundary condition set (force on a node)

Assign boundary condition to a node

Implementation of function to print out model data

```
 void MyStructObject::print_model() {

  std::cout << "Node list:\n";
  unsigned index = 0;
  for ( std::vector<node_type>::const_iterator nit = Nodes.begin(); nit
!= Nodes.end(); ++nit, ++index ) {
    std::cout << index << " " << *nit << "\n";
  }

  std::cout << "Element list:\n";
  index = 0;
  for ( PtrVector<element_type*>::const_iterator eit =
Elements.begin(); eit != Elements.end(); ++eit, ++index ) {
    std::cout << index << " " << **eit << "\n";
  }

  std::cout << "Boundary condition list:\n";
  index=0;
  for ( std::vector<bc_type>::const_iterator bit =
BoundaryConditions.begin(); bit!=BoundaryConditions.end(); ++bit,
++index){
    std::cout << index << " " << *bit << "\n";
  }

}
```

Implementation of function to store results in a txt-file

```
void MyStructObject::Write_Solution(std::string filename, std::string
path) {

  ofstream RectangleSolution ((path + "/" + filename).c_str());

  mtl::Dense_Vector vector;

  if (RectangleSolution) {
    RectangleSolution << "Solution Rectangle FELyX Model" << "\n";
    for (std::vector<node_type>::iterator nit = Nodes.begin(); nit <
Nodes.end(); ++nit) {
      vector = (*nit).GetStresses();
      RectangleSolution << " Node x-Coord " << (*nit).GetCx() << " y-
Coord" << (*nit).GetCy();
      for ( unsigned i = 0; i < vector.size(); ++i){
        RectangleSolution << " " << vector[i];
      }
      RectangleSolution << "\n";
    }
  }


  RectangleSolution.close();
}
```

Create new file

Write data in the txt-file

Close txt-file

End of function create_model

MyStructObjectApp.cpp

- Call the functions and solve the FE model

```
//
// Application to instanciate and call MyStructObject
//

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include "MyStructObject.h"

int main( int ac, char* av[] ) {

  try {

    std::cout << "FE analysis with MyStructObject\n";

    MyStructObject FEM;

    FEM.create_model();

    FEM.print_model();

    FEM.SaveAnsysModel("Rectangle.ansys", "./");

    FEM.SparseSolver();

    FEM.EvalStresses();

    FEM.PrintGlobalStatus();

    FEM.Write_Solution("RectangleSolution.txt", "./");


  }

  // Catch exceptions
  catch ( std::exception & e ) {
    cerr << e.what() << endl;
    return 1;
  }

  std::cout << "FELyX analysis done\n";

  return 0;
}
```

Application to instanciate and call MyStructObject

Include header files

Include main header

Define namespace

Instanciate MyStructObject

Create model

Print model

Export as ANSYS file

Solve FE problem

Eval stresses

Print global status

Write results in a txt-file