```
//*****************************************************************//
//    Albany 2.0:  Copyright 2012 Sandia Corporation               //
//    This Software is released under the BSD license detailed     //
//    in the file "license.txt" in the top-level Albany directory  //
//*****************************************************************//

#ifndef HMCPROBLEM_HPP
#define HMCPROBLEM_HPP

#include "Teuchos_RCP.hpp"
#include "Teuchos_ParameterList.hpp"

#include "Albany_AbstractProblem.hpp"

#include "Phalanx.hpp"
#include "PHAL_Workset.hpp"
#include "PHAL_Dimension.hpp"
#include "PHAL_AlbanyTraits.hpp"

// To do:
// --  Add multiblock support (See mechanics example problem)
// --  Add density as input.  Currently hardwired to implicit value of 1.0.
// --  Add Currant limit.  Newmark integrator only seems to work for beta=0.25.
// --  Add artificial viscosity.
// --  Add hourglass stabilization for single point integration.
```

This source has been annotated with latex comments. Use the eqcc script to compile into a summary pdf. The source is best viewed using folding in vim (i.e.,

```
:g/\\begin{text}/foldc
```

)

```
namespace Albany {

  /*!
   * \brief Abstract interface for representing a 2-D finite element
   * problem.
   */
  class HMCProblem : public Albany::AbstractProblem {
  public:

    //! Default constructor
    HMCProblem(
      const Teuchos::RCP<Teuchos::ParameterList>& params_,
      const Teuchos::RCP<ParamLib>& paramLib_,
      const int numDim_,
                      const Teuchos::RCP<const Epetra_Comm>& comm);


    //! Destructor
    virtual ~HMCProblem();

    //! Return number of spatial dimensions
    virtual int spatialDimension() const { return numDim; }

    //! Build the PDE instantiations, boundary conditions, and initial solution
    virtual void buildProblem(
      Teuchos::ArrayRCP<Teuchos::RCP<Albany::MeshSpecsStruct> >  meshSpecs,
      StateManager& stateMgr);

    // Build evaluators
    virtual Teuchos::Array< Teuchos::RCP<const PHX::FieldTag> >
    buildEvaluators(
      PHX::FieldManager<PHAL::AlbanyTraits>& fm0,
      const Albany::MeshSpecsStruct& meshSpecs,
      Albany::StateManager& stateMgr,
      Albany::FieldManagerChoice fmchoice,
```

```
    const Teuchos::RCP<Teuchos::ParameterList>& responseList);

  //! Each problem must generate it's list of valid parameters
  Teuchos::RCP<const Teuchos::ParameterList> getValidProblemParameters() const;

  void getAllocatedStates(Teuchos::ArrayRCP<Teuchos::ArrayRCP<Teuchos::RCP<Intrepid::FieldContainer<RealType> > > > oldState_,
  Teuchos::ArrayRCP<Teuchos::ArrayRCP<Teuchos::RCP<Intrepid::FieldContainer<RealType> > > > newState_
  ) const;

private:

  //! Private to prohibit copying
  HMCProblem(const HMCProblem&);

  //! Private to prohibit copying
  HMCProblem& operator=(const HMCProblem&);


  void parseMaterialModel(Teuchos::RCP<Teuchos::ParameterList>& p,
                     const Teuchos::RCP<Teuchos::ParameterList>& params) const;

  Teuchos::RCP<QCAD::MaterialDatabase> material_db_;


public:

  //! Main problem setup routine. Not directly called, but indirectly by following functions
  template <typename EvalT>
  Teuchos::RCP<const PHX::FieldTag>
  constructEvaluators(
    PHX::FieldManager<PHAL::AlbanyTraits>& fm0,
    const Albany::MeshSpecsStruct& meshSpecs,
    Albany::StateManager& stateMgr,
    Albany::FieldManagerChoice fmchoice,
    const Teuchos::RCP<Teuchos::ParameterList>& responseList);
```

```
    void constructDirichletEvaluators(const Albany::MeshSpecsStruct& meshSpecs);
    void constructNeumannEvaluators(const Teuchos::RCP<Albany::MeshSpecsStruct>& meshSpecs);


  protected:

    //! Boundary conditions on source term
    bool haveSource;
    int numDim;
    int numMicroScales;

    std::string matModel;
    Teuchos::RCP<Albany::Layouts> dl;

    Teuchos::ArrayRCP<Teuchos::ArrayRCP<Teuchos::RCP<Intrepid::FieldContainer<RealType> > > > oldState;
    Teuchos::ArrayRCP<Teuchos::ArrayRCP<Teuchos::RCP<Intrepid::FieldContainer<RealType> > > > newState;
  };

}

#include "Albany_SolutionAverageResponseFunction.hpp"
#include "Albany_SolutionTwoNormResponseFunction.hpp"
#include "Albany_SolutionMaxValueResponseFunction.hpp"
#include "Albany_Utils.hpp"
#include "Albany_ProblemUtils.hpp"
#include "Albany_ResponseUtilities.hpp"
#include "Albany_EvaluatorUtils.hpp"
#include "HMC_StrainDifference.hpp"
#include "HMC_TotalStress.hpp"
#include "FieldNameMap.hpp"

#include "Strain.hpp"
#include "DefGrad.hpp"
#include "HMC_Stresses.hpp"
#include "PHAL_SaveStateField.hpp"
#include "ElasticityResid.hpp"
```

```
#include "HMC_MicroResidual.hpp"

#include "Time.hpp"
#include "ConstitutiveModelParameters.hpp"
#include "ConstitutiveModelInterface.hpp"


#include <sstream>

template <typename EvalT>
Teuchos::RCP<const PHX::FieldTag>
Albany::HMCProblem::constructEvaluators(
  PHX::FieldManager<PHAL::AlbanyTraits>& fm0,
  const Albany::MeshSpecsStruct& meshSpecs,
  Albany::StateManager& stateMgr,
  Albany::FieldManagerChoice fieldManagerChoice,
  const Teuchos::RCP<Teuchos::ParameterList>& responseList)
{
   using Teuchos::RCP;
   using Teuchos::rcp;
   using Teuchos::ParameterList;
   using PHX::DataLayout;
   using PHX::MDALayout;
   using std::vector;
   using PHAL::AlbanyTraits;

  // get the name of the current element block
   std::string eb_name = meshSpecs.ebName;

  // get the name of the material model to be used (and make sure there is one)
  std::string material_model_name =
      material_db_->
          getElementBlockSublist(eb_name, "Material Model").get<std::string>(
          "Model Name");
  TEUCHOS_TEST_FOR_EXCEPTION(material_model_name.length() == 0, std::logic_error,
      "A material model must be defined for block: "
```

```
          + eb_name);

#ifdef ALBANY_VERBOSE
  *out << "In MechanicsProblem::constructEvaluators" << std::endl;
  *out << "element block name: " << eb_name << std::endl;
  *out << "material model name: " << material_model_name << std::endl;
#endif

   RCP<shards::CellTopology> cellType = rcp(new shards::CellTopology (&meshSpecs.ctd));
   RCP<Intrepid::Basis<RealType, Intrepid::FieldContainer<RealType> > >
     intrepidBasis = Albany::getIntrepidBasis(meshSpecs.ctd);

   const int numNodes = intrepidBasis->getCardinality();
   const int worksetSize = meshSpecs.worksetSize;

   Intrepid::DefaultCubatureFactory<RealType> cubFactory;
   RCP <Intrepid::Cubature<RealType> > cubature = cubFactory.create(*cellType, meshSpecs.cubatureDegree);

   const int numDim = cubature->getDimension();
   const int numQPts = cubature->getNumPoints();
   const int numVertices = cellType->getNodeCount();

   *out << "Field Dimensions: Workset=" << worksetSize
        << ", Vertices= " << numVertices
        << ", Nodes= " << numNodes
        << ", QuadPts= " << numQPts
        << ", Dim= " << numDim << std::endl;


   // Construct standard FEM evaluators with standard field names
   dl = rcp(new Albany::Layouts(worksetSize,numVertices,numNodes,numQPts,numDim));
   TEUCHOS_TEST_FOR_EXCEPTION(dl->vectorAndGradientLayoutsAreEquivalent==false, std::logic_error,
                              "Data Layout Usage in Mechanics problems assume vecDim = numDim");

   Albany::EvaluatorUtils<EvalT, PHAL::AlbanyTraits> evalUtils(dl);
```

```
const int numMacroScales = 1;

// Define Field Names
Teuchos::ArrayRCP<std::string> macro_dof_names(numMacroScales);
macro_dof_names[0] = "Displacement";
Teuchos::ArrayRCP<std::string> macro_resid_names(numMacroScales);
macro_resid_names[0] = macro_dof_names[0] + " Residual";

Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_dof_names(numMicroScales);
Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_resid_names(numMicroScales);
Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_scatter_names(numMicroScales);
for(int i=0;i<numMicroScales;i++){
   micro_dof_names[i].resize(1);
   micro_resid_names[i].resize(1);
   micro_scatter_names[i].resize(1);
   std::stringstream dofname;
   dofname << "Microstrain_" << i;
   micro_dof_names[i][0] = dofname.str();
   micro_resid_names[i][0] = dofname.str() + " Residual";
   micro_scatter_names[i][0] = dofname.str() + " Scatter";
 }

Teuchos::ArrayRCP<std::string> macro_dof_names_dotdot(numMacroScales);
Teuchos::ArrayRCP<std::string> macro_resid_names_dotdot(numMacroScales);
Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_dof_names_dotdot(numMicroScales);
Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_resid_names_dotdot(numMicroScales);
Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_scatter_names_dotdot(numMicroScales);
macro_dof_names_dotdot[0] = macro_dof_names[0]+"_dotdot";
macro_resid_names_dotdot[0] = macro_resid_names[0]+" Residual";
for(int i=0;i<numMicroScales;i++){
  micro_dof_names_dotdot[i].resize(1);
  micro_resid_names_dotdot[i].resize(1);
  micro_scatter_names_dotdot[i].resize(1);
  micro_dof_names_dotdot[i][0] = micro_dof_names[i][0]+"_dotdot";
  micro_resid_names_dotdot[i][0] = micro_resid_names_dotdot[i][0]+" Residual";
  micro_scatter_names_dotdot[i][0] = micro_scatter_names_dotdot[i][0]+" Scatter";
```

```
  }

// Gather Solution (displacement and acceleration)
```

Gather solution data from solver data structures to grid based structures.
DEPENDENT FIELDS:
None.
EVALUATED FIELDS:

u_{Ii}	Nodal displacements	"Displacement"	dims(cell,I=nNodes,i=vecDim)
a_{Ii}	Nodal accelerations	"Displacement_dotdot"	dims(cell,I=nNodes,i=vecDim)

```
  int vectorRank = 1;
  fm0.template registerEvaluator<EvalT>
    (evalUtils.constructGatherSolutionEvaluator_withAcceleration(vectorRank, macro_dof_names, Teuchos::null, macro_dof_names_dotdot));

// Gather Solution (microstrains and micro accelerations)
```

Gather solution data from solver data structures to grid based structures.
DEPENDENT FIELDS:
None.
EVALUATED FIELDS:

ϵ^n_{Iij}	Nodal microstrains at scale 'n'	"Microstrain_n"	dims(cell,I=nNodes,i=vecDim,j=vecDim)
$\ddot{\epsilon}^n_{Iij}$	Nodal micro accelerations at scale 'n'	"Microstrain_n_dotdot"	dims(cell,I=nNodes,i=vecDim,j=vecDim)

```
  int dof_offset = numDim; // dof layout is {x, y, ..., xx, xy, xz, yx, ...}
  int dof_stride = numDim*numDim;
  int tensorRank = 2;
  for(int i=0;i<numMicroScales;i++){
    fm0.template registerEvaluator<EvalT>
      (evalUtils.constructGatherSolutionEvaluator_withAcceleration(
         tensorRank, micro_dof_names[i], Teuchos::null, micro_dof_names_dotdot[i], dof_offset+i*dof_stride) );
```

```
  }

// Gather Coordinates
```

Gather coordinate data from solver data structures to grid based structures.
DEPENDENT FIELDS:
None.
EVALUATED FIELDS:

x_{Ii}	Nodal coordinates	"Coord Vec"	dims(cell,I=nNodes,i=vecDim)

```
  fm0.template registerEvaluator<EvalT>
    (evalUtils.constructGatherCoordinateVectorEvaluator());

// Compute gradient matrix and weighted basis function values in current coordinates
```

Register new evaluator.
DEPENDENT FIELDS:

x_{Ii}	Nodal coordinates	"Coord Vec"	dims(cell,I=nNodes,i=vecDim)

EVALUATED FIELDS:

$det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted measure	"Weights"	dims(cell,p=nQPs)
$det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)$	Jacobian determinant	Jacobian Det"	dims(cell,p=nQPs)
$N_I(\mathbf{x}_p)$	Basis function values	"BF"	dims(cell,I=nNodes,p=nQPs)
$N_I(\mathbf{x}_p)\ det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted basis function values	"wBF"	dims(cell,I=nNode,p=nQPs)
$\frac{\partial N_I(\mathbf{x}_p)}{\partial \xi_k} J^{-1}_{kj}$	Gradient matrix wrt physical frame	"Grad BF"	dims(cell,I=nNodes,p=nQPs,j=spcDim)
$\frac{\partial N_I(\mathbf{x}_p)}{\partial \xi_k} J^{-1}_{kj} det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted gradient matrix wrt current config	"wGrad BF"	dims(cell,I=nNodes,p=nQPs,j=spcDim)

```
  fm0.template registerEvaluator<EvalT>
    (evalUtils.constructComputeBasisFunctionsEvaluator(cellType, intrepidBasis, cubature));
```

```
// Project displacements to Gauss points
```

Register new evaluator:

$$u_i(\xi_p) = N_I(\xi_p)u_{Ii}$$
$$(c,p,i) = (c,I,p) * (c,I,i)$$

DEPENDENT FIELDS:

u_{Ii}	Nodal Displacements	"Displacements"	dims(cell,I=nNodes,i=vecDim)
$N_I(\xi_p)$	Basis Functions	"BF"	dims(cell,I=nNodes,p=nQPs)

EVALUATED FIELDS:

$u_i(\xi_p)$	Displacements at quadrature points	"Displacements"	dims(cell,p=nQPs,i=vecDim)

```
  fm0.template registerEvaluator<EvalT>
    (evalUtils.constructDOFVecInterpolationEvaluator(macro_dof_names[0]));

// Project microstrains to Gauss points
```

Register new evaluator:

$$\epsilon^n_{ij}(\xi_p) = N_I(\xi_p)\epsilon^n_{ijI}$$
$$(c,p,i,j) = (c,I,p) * (c,I,i,j)$$

DEPENDENT FIELDS:

ϵ^n_{Iij}	Nodal microstrains at scale 'n'	"Microstrain_n"	dims(cell,I=nNodes,i=vecDim,j=vecDim)
$N_I(\xi_p)$	Basis Functions	"BF"	dims(cell,I=nNodes,p=nQPs)

EVALUATED FIELDS:

$\epsilon^n_{ij}(\xi_p)$	Microstrains at scale 'n' at quadrature points	"Microstrain_n"	dims(cell,p=nQPs,i=vecDim,j=spcDim)

```
  for(int i=0;i<numMicroScales;i++)
    fm0.template registerEvaluator<EvalT>
```

```
      (evalUtils.constructDOFTensorInterpolationEvaluator(micro_dof_names[i][0],
       dof_offset+i*dof_stride));

// Project accelerations to Gauss points
```

Register new evaluator:

$$a_i(\xi_p) = N_I(\xi_p) a_{Ii}$$
$$(c,p,i) = (c,I,p) * (c,I,i)$$

DEPENDENT FIELDS:

a_{Ii}	Nodal Acceleration	"Displacement_dotdot"	dims(cell,I=nNodes,i=vecDim)
$N_I(\xi_p)$	Basis Functions	"BF"	dims(cell,I=nNodes,p=nQPs)

EVALUATED FIELDS:

$a_i(\xi_p)$	Acceleration at quadrature points	"Displacement_dotdot"	dims(cell,p=nQPs,i=vecDim)

```
   fm0.template registerEvaluator<EvalT>
     (evalUtils.constructDOFVecInterpolationEvaluator(macro_dof_names_dotdot[0]));

// Project micro accelerations to Gauss points
```

Register new evaluator:

$$\ddot{\epsilon}^n_{ij}(\xi_p) = N_I(\xi_p) \ddot{\epsilon}^n_{Iij}$$
$$(c,p,i,j) = (c,I,p) * (c,I,i,j)$$

DEPENDENT FIELDS:

$\ddot{\epsilon}^n_{Iij}$	Nodal micro acceleration at scale 'n'	"Microstrain_n_dotdot"	dims(cell,I=nNodes,i=vecDim,j=vecDim)
$N_I(\xi_p)$	Basis Functions	"BF"	dims(cell,I=nNodes,p=nQPs)

EVALUATED FIELDS:

$\ddot{\epsilon}^n_{ij}(\xi_p)$	Micro acceleration at scale 'n' at quadrature points	"Microstrain_n_dotdot"	dims(cell,p=nQPs,i=vecDim,j=vecDim)

```
  for(int i=0;i<numMicroScales;i++)
    fm0.template registerEvaluator<EvalT>
      (evalUtils.constructDOFTensorInterpolationEvaluator(micro_dof_names_dotdot[i][0],
       dof_offset+i*dof_stride));

// Project nodal coordinates to Gauss points
```

Register new evaluator: Compute Gauss point locations from nodal locations.

$$x_{pi} = N_I(\xi_p) x_{Ii}$$
$$(c,p,i) = (c,I,p) * (c,I,i)$$

DEPENDENT FIELDS:

x_{Ii}	Nodal coordinates	"Coord Vec"	dims(cell,I=nNodes,i=vecDim)

EVALUATED FIELDS:

x_{pi}	Gauss point coordinates	"Coord Vec"	dims(cell,p=nQPs,i=vecDim)

```
  fm0.template registerEvaluator<EvalT>
    (evalUtils.constructMapToPhysicalFrameEvaluator(cellType, cubature));

// Compute displacement gradient
```

New evaluator:

$$\left.\frac{\partial u_i}{\partial x_j}\right|_{\xi_p} = \partial_j N_I(\xi_p) u_{iI}$$
$$(c,p,i,j) = (c,I,p,j) * (c,I,i)$$

DEPENDENT FIELDS:

u_{Ii}	Nodal Displacement	"Displacement"	dims(cell,I=nNodes,i=vecDim)
$B_I(\xi_p)$	Gradient of Basis Functions	"Grad BF"	dims(cell,I=nNodes,p=nQPs,i=vecDim)

EVALUATED FIELDS:

$\left.\frac{\partial u_i}{\partial x_j}\right\|_{\xi_p}$	Gradient of node vector	"Displacement Gradient"	dims(cell,p=nQPs,i=vecDim,j=spcDim)

```
  fm0.template registerEvaluator<EvalT>
    (evalUtils.constructDOFVecGradInterpolationEvaluator(macro_dof_names[0]));

// Compute microstrain gradient
```

Register new evaluator:

$$\left.\frac{\partial \epsilon^n_{ij}}{\partial x_k}\right|_{\xi_p} = \partial_k N_I(\xi_p)\epsilon^n_{Iij}$$
$$(c,p,i,j,k) = (c,I,p,k) * (c,I,i,j)$$

DEPENDENT FIELDS:

ϵ^n_{Iij}	Nodal microstrain at scale 'n'	"Microstrain_n"	dims(cell,I=nNodes,i=vecDim,j=vecDim)
$B_I(\xi_p)$	Gradient of Basis Functions	"Grad BF"	dims(cell,I=nNodes,p=nQPs,i=vecDim)

EVALUATED FIELDS:

$\left.\frac{\partial \epsilon^n_{ij}}{\partial x_k}\right\|_{\xi_p}$	Microstrain gradient at scale 'n'	"Microstrain_n Gradient"	dims(cell,p=nQPs,i=vecDim,j=vecDim,k=spcDim)

```
  for(int i=0;i<numMicroScales;i++)
    fm0.template registerEvaluator<EvalT>
      (evalUtils.constructDOFTensorGradInterpolationEvaluator(micro_dof_names[i][0],dof_offset+i*dof_stride));

  // Temporary variable used numerous times below
  Teuchos::RCP<PHX::Evaluator<AlbanyTraits> > ev;

// Compute strain
```

New evaluator:

$$\epsilon^p_{ij} = \frac{1}{2}\left(\left.\frac{\partial u_i}{\partial x_j}\right|_{\xi_p} + \left.\frac{\partial u_j}{\partial x_i}\right|_{\xi_p}\right)$$
$$(c,p,i,j) = ((c,p,i,j) + (c,p,j,i)/2.0)$$

DEPENDENT FIELDS:

$\left.\frac{\partial u_i}{\partial x_j}\right\|_{\xi_p}$	Gradient of displacement	"Displacement Gradient"	dims(cell, p=nQPs, i=vecDim, j=spcDim)

EVALUATED FIELDS:

ϵ_{ij}^{p}	Infinitesimal strain	"Strain"	dims(cell, p=nQPs, i=vecDim, j=spcDim)

```
  {
    RCP<ParameterList> p = rcp(new ParameterList("Strain"));

    //Input
    p->set<std::string>("Gradient QP Variable Name", "Displacement Gradient");

    //Output
    p->set<std::string>("Strain Name", "Strain");

    ev = rcp(new LCM::Strain<EvalT,AlbanyTraits>(*p,dl));
    fm0.template registerEvaluator<EvalT>(ev);
  }

// Compute microstrain difference
```

Register new evaluator:

$$\Delta\epsilon_{ij}^{np} = \epsilon_{ij}^{p} - \epsilon_{ij}^{np}$$

DEPENDENT FIELDS:

ϵ_{ij}^{p}	Macro strain	"Strain"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
ϵ_{ijI}^{n}	Nodal microstrains at scale 'n'	"Microstrain_1"	dims(cell, I=nNodes, i=vecDim, j=vecDim)

EVALUATED FIELDS:

$\Delta\epsilon_{ij}^{np}$	Strain Difference at scale 'n'	"Strain Difference n"	dims(cell, p=nQPs, i=vecDim, j=spcDim)

```
  for(int i=0;i<numMicroScales;i++){
```

```
    RCP<ParameterList> p = rcp(new ParameterList("Strain Difference"));

    //Input
    p->set<std::string>("Micro Strain Name", micro_dof_names[i][0]);
    p->set<std::string>("Macro Strain Name", "Strain");

    //Output
    std::stringstream sd;
    sd << "Strain Difference " << i;
    p->set<std::string>("Strain Difference Name", sd.str());

    ev = rcp(new HMC::StrainDifference<EvalT,AlbanyTraits>(*p,dl));
    fm0.template registerEvaluator<EvalT>(ev);
  }


  { // Constitutive Model Parameters
    RCP<ParameterList> p = rcp(
        new ParameterList("Constitutive Model Parameters"));
    std::string matName = material_db_->getElementBlockParam<std::string>(
        eb_name, "material");
    Teuchos::ParameterList& param_list =
        material_db_->getElementBlockSublist(eb_name, matName);

    // pass through material properties
    p->set<Teuchos::ParameterList*>("Material Parameters", &param_list);

    RCP<LCM::ConstitutiveModelParameters<EvalT, AlbanyTraits> > cmpEv =
        rcp(new LCM::ConstitutiveModelParameters<EvalT, AlbanyTraits>(*p, dl));
    fm0.template registerEvaluator<EvalT>(cmpEv);
  }

// Compute stresses
```

Register new evaluator:

$$\{\sigma^p_{ij}, \bar{\beta}^{np}_{ij}, \bar{\bar{\beta}}^{np}_{ijk}\} = f(\{\epsilon^p_{ij}, \Delta\epsilon^{np}_{ij}, \epsilon^{np}_{ij,k}\})$$

DEPENDENT FIELDS:

ϵ^p_{ij}	Macro strain	"Strain"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
$\Delta\epsilon^{np}_{ij}$	Strain Difference 'n'	"Strain Difference 1"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
$\epsilon^{np}_{ij,k}$	Microstrain gradient 'n'	"Microstrain_n Gradient"	dims(cell, p=nQPs, i=vecDim, j=vecDim, k=spcDim)

EVALUATED FIELDS:

σ^p_{ij}	Macro Stress	"Stress"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
$\bar{beta}^{np}_{ij}$	Micro stress 'n'	"Micro Stress n"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
$\bar{\bar{beta}}^{np}_{ijk}$	Double stress 'n'	"Double Stress n"	dims(cell, p=nQPs, i=vecDim, j=spcDim, k=spcDim)

```
{
  RCP<ParameterList> p = rcp(new ParameterList("Constitutive Model Interface"));
  std::string matName = material_db_->getElementBlockParam<std::string>(eb_name, "material");
  Teuchos::ParameterList& param_list = material_db_->getElementBlockSublist(eb_name, matName);

  // construct field name map
  // required
  LCM::FieldNameMap
  field_name_map(false);
  RCP<std::map<std::string, std::string> >
  fnm = field_name_map.getMap();
  param_list.set<RCP<std::map<std::string, std::string> > >("Name Map", fnm);
  p->set<Teuchos::ParameterList*>("Material Parameters", &param_list);
  // end required

  p->set<Teuchos::ParameterList*>("Material Parameters", &param_list);

  RCP<LCM::ConstitutiveModelInterface<EvalT, AlbanyTraits> > cmiEv =
      rcp(new LCM::ConstitutiveModelInterface<EvalT, AlbanyTraits>(*p, dl));
  fm0.template registerEvaluator<EvalT>(cmiEv);

  // register state variables
```

```
    for (int sv(0); sv < cmiEv->getNumStateVars(); ++sv) {
      cmiEv->fillStateVariableStruct(sv);
      p = stateMgr.registerStateVariable(cmiEv->getName(),
          cmiEv->getLayout(),
          dl->dummy,
          eb_name,
          cmiEv->getInitType(),
          cmiEv->getInitValue(),
          cmiEv->getStateFlag(),
          cmiEv->getOutputFlag());
      ev = rcp(new PHAL::SaveStateField<EvalT, AlbanyTraits>(*p));
      fm0.template registerEvaluator<EvalT>(ev);
    }
  }

// Compute total stress
```

Register new evaluator:

$$\sigma^{tp}_{ij} = \sigma^{p}_{ij} + \sum_n \bar{\beta}^{np}_{ij}$$

DEPENDENT FIELDS:

σ^{tp}_{ij}	Total stress	"Total Stress"	dims(cell, p=nQPs, i=vecDim, j=spcDim)

EVALUATED FIELDS:

σ^{p}_{ij}	Macro Stress	"Stress"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
$beta^{np}_{ij}$	Micro stress 'n'	"Micro Stress n"	dims(cell, p=nQPs, i=vecDim, j=spcDim)

```
  {
    RCP<ParameterList> p = rcp(new ParameterList("Total Stress"));

    p->set<int>("Additional Scales", numMicroScales);

    //Input
    p->set<std::string>("Macro Stress Name", "Stress");
```

```
    p->set< RCP<DataLayout> >("QP 2Tensor Data Layout", dl->qp_tensor);
    for(int i=0;i<numMicroScales;i++){
      std::string ms = Albany::strint("Micro Stress",i);
      std::string msname(ms); msname += " Name";
      p->set<std::string>(msname, ms);
    }
    //Output
    p->set<std::string>("Total Stress Name", "Total Stress");

    ev = rcp(new HMC::TotalStress<EvalT,AlbanyTraits>(*p,dl));
    fm0.template registerEvaluator<EvalT>(ev);
  }


// Compute macro residual
```

Register new evaluator:

$$f_{Ii} = \sum_p \frac{\partial N_I(\mathbf{x}_p)}{\partial \xi_k} J^{-1}_{kj} det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p \sigma^{tp}_{ij} + \sum_p N_I(\mathbf{x}_p)\ det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p a^p_i$$

DEPENDENT FIELDS:

σ^{tp}_{ij}	Total stress	"Total Stress"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
$\frac{\partial N_I(\mathbf{x}_p)}{\partial \xi_k} J^{-1}_{kj} det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted Gradient matrix wrt current config	"wGrad BF"	dims(cell, I=nNodes, p=nQPs, i=spcDim)
a^p_i	Acceleration at quadrature points	"Displacement_dotdot"	dims(cell, p=nQPs, i=vecDim)
$N_I(\mathbf{x}_p)\ det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted BF	"wBF"	dims(cell, I=nNodes, p=nQPs)

EVALUATED FIELDS:

f_{Ii}	Macroscale Residual	"Displacement Residual"	dims(cell, I=nNodes, i=spcDim)

```
  {
    RCP<ParameterList> p = rcp(new ParameterList("Displacement Resid"));

    //Input
```

```
    p->set<std::string>("Stress Name", "Total Stress");
    p->set< RCP<DataLayout> >("QP Tensor Data Layout", dl->qp_tensor);

    p->set<std::string>("Weighted Gradient BF Name", "wGrad BF");
    p->set< RCP<DataLayout> >("Node QP Vector Data Layout", dl->node_qp_vector);

    // extra input for time dependent term
    p->set<std::string>("Weighted BF Name", "wBF");
    p->set< RCP<DataLayout> >("Node QP Scalar Data Layout", dl->node_qp_scalar);
    p->set<std::string>("Time Dependent Variable Name", macro_dof_names_dotdot[0]);
    p->set< RCP<DataLayout> >("QP Vector Data Layout", dl->qp_vector);

    //Output
    p->set<std::string>("Residual Name", macro_resid_names[0]);
    p->set< RCP<DataLayout> >("Node Vector Data Layout", dl->node_vector);

    ev = rcp(new LCM::ElasticityResid<EvalT,AlbanyTraits>(*p));
    fm0.template registerEvaluator<EvalT>(ev);
  }
// Compute micro residuals
```

Register new evaluator:

$$f^n_{Iij} = \sum_p \frac{\partial N_I(\mathbf{x}_p)}{\partial \xi_l} J^{-1}_{lk} \bar{\bar{\beta}}^{np}_{ijk} \} det \left(\frac{\partial x_{ip}}{\partial \xi_j} \right) \omega_p + \sum_p N_I(\mathbf{x}_p) \bar{\beta}^{np}_{ij} \det \left(\frac{\partial x_{ip}}{\partial \xi_j} \right) \omega_p + \sum_p N_I(\mathbf{x}_p) \ddot{\epsilon}^{np}_{ij} \det \left(\frac{\partial x_{ip}}{\partial \xi_j} \right) \omega_p$$

DEPENDENT FIELDS:

$\bar{beta}^{np}_{ij}$	Micro stress 'n'	"Micro Stress n"	dims(cell, p=nQPs, i=vecDim, j=spcDim)
$\bar{\bar{beta}}^{np}_{ijk}$	Double stress 'n'	"Double Stress n"	dims(cell, p=nQPs, i=vecDim, j=spcDim, k=spcDim)
$\ddot{\epsilon}^n_{Iij}$	micro accel at scale 'n'	"Microstrain_n_dotdot"	dims(cell,I=nNodes,i=vecDim,j=vecDim)
$\frac{\partial N_I(\mathbf{x}_p)}{\partial \xi_k} J^{-1}_{kj} det \left(\frac{\partial x_{ip}}{\partial \xi_j} \right) \omega_p$	Weighted Gradient matrix wrt current config	"wGrad BF"	dims(cell, I=nNodes, p=nQPs, i=spcDim)
$N_I(\mathbf{x}_p)\ det \left(\frac{\partial x_{ip}}{\partial \xi_j} \right) \omega_p$	Weighted BF	"wBF"	dims(cell, I=nNodes, p=nQPs)

EVALUATED FIELDS:

f^n_{Iij}	Residual at scale 'n'	"Microstrain_n Residual"	dims(cell, I=nNodes, i=vecDim, j=vecDim)

```
  for(int i=0;i<numMicroScales;i++){
    RCP<ParameterList> p = rcp(new ParameterList("Microstrain Resid"));

    //Input: Micro stresses
    std::string ms = Albany::strint("Micro Stress",i);
    p->set<std::string>("Micro Stress Name", ms);
    p->set< RCP<DataLayout> >("QP Tensor Data Layout", dl->qp_tensor);

    std::string ds = Albany::strint("Double Stress",i);
    p->set<std::string>("Double Stress Name", ds);
    p->set< RCP<DataLayout> >("QP 3Tensor Data Layout", dl->qp_tensor3);

    p->set<std::string>("Weighted Gradient BF Name", "wGrad BF");
    p->set< RCP<DataLayout> >("Node QP Vector Data Layout", dl->node_qp_vector);

    p->set<std::string>("Weighted BF Name", "wBF");
    p->set< RCP<DataLayout> >("Node QP Scalar Data Layout", dl->node_qp_scalar);

    // extra input for time dependent term
    p->set<std::string>("Time Dependent Variable Name", micro_dof_names[i][0]+"_dotdot");
    p->set< RCP<DataLayout> >("QP Vector Data Layout", dl->qp_vector);

    //Output
    p->set<std::string>("Residual Name", micro_resid_names[i][0]);
    p->set< RCP<DataLayout> >("Node Tensor Data Layout", dl->node_tensor);

    ev = rcp(new HMC::MicroResidual<EvalT,AlbanyTraits>(*p));
    fm0.template registerEvaluator<EvalT>(ev);
  }

// Scatter macroscale forces
```

Register new evaluator: Scatter the nodal forces from grid based structures to solver data structures.
DEPENDENT FIELDS:
f_{Ii} Macroscale Residual "Displacement Residual" dims(cell, I=nNodes, i=vecDim)
EVALUATED FIELDS:
None.

```
   fm0.template registerEvaluator<EvalT>
     (evalUtils.constructScatterResidualEvaluator(vectorRank, macro_resid_names));

// Scatter microscale forces
```

Register new evaluator: Scatter the nodal forces from grid based structures to solver data structures.
DEPENDENT FIELDS:
f^n_{Iij} Microscale Residual "Microstrain_n Residual" dims(cell, I=nNodes, i=vecDim, j=vecDim)
EVALUATED FIELDS:
None.

```
  int numTensorFields = numDim*numDim;
  int dofOffset = numDim;
  for(int i=0;i<numMicroScales;i++){ // Micro forces
    fm0.template registerEvaluator<EvalT>
      (evalUtils.constructScatterResidualEvaluator(tensorRank, micro_resid_names[i], dofOffset, micro_scatter_names[i][0]));
    dofOffset += numTensorFields;
  }

  { // Time
    RCP<ParameterList> p = rcp(new ParameterList);

    p->set<std::string>("Time Name", "Time");
    p->set<std::string>("Delta Time Name", "Delta Time");
    p->set< RCP<DataLayout> >("Workset Scalar Data Layout", dl->workset_scalar);
```

```
    p->set<RCP<ParamLib> >("Parameter Library", paramLib);
    p->set<bool>("Disable Transient", true);

    ev = rcp(new LCM::Time<EvalT,AlbanyTraits>(*p));
    fm0.template registerEvaluator<EvalT>(ev);
    p = stateMgr.registerStateVariable("Time",dl->workset_scalar, dl->dummy, eb_name, "scalar", 0.0, true);
    ev = rcp(new PHAL::SaveStateField<EvalT,AlbanyTraits>(*p));
    fm0.template registerEvaluator<EvalT>(ev);
  }


  if (fieldManagerChoice == Albany::BUILD_RESID_FM)  {
    PHX::Tag<typename EvalT::ScalarT> res_tag("Scatter", dl->dummy);
    fm0.requireField<EvalT>(res_tag);
    for(int i=0;i<numMicroScales;i++){ // Micro forces
      PHX::Tag<typename EvalT::ScalarT> res_tag(micro_scatter_names[i][0], dl->dummy);
      fm0.requireField<EvalT>(res_tag);
    }
    return res_tag.clone();
  }
  else if (fieldManagerChoice == Albany::BUILD_RESPONSE_FM) {
    Albany::ResponseUtilities<EvalT, PHAL::AlbanyTraits> respUtils(dl);
    return respUtils.constructResponses(fm0, *responseList, stateMgr);
  }

  return Teuchos::null;
}

#endif // ALBANY_ELASTICITYPROBLEM_HPP
```