```
//*****************************************************************//
//    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"
```

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_);

  //! 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;

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_EvaluatorUtils.hpp"
#include "HMC_StrainDifference.hpp"

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

#include "Time.hpp"
#include "CapExplicit.hpp"
#include "CapImplicit.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 elementBlockName = meshSpecs.ebName;

   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::HMCEvaluatorUtils<EvalT, PHAL::AlbanyTraits> evalUtilsHMC(dl);
Albany::EvaluatorUtils<EvalT, PHAL::AlbanyTraits> evalUtils(dl);
bool supportsTransient=true;

const int numMacroScales = 1;

// Define Field Names
Teuchos::ArrayRCP<std::string> macro_dof_names(1);
macro_dof_names[0] = "Displacement";

Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_dof_names(numMicroScales);
  for(int i=0;i<numMicroScales;i++){
    micro_dof_names[i].resize(1);
    std::stringstream dofname;
    dofname << "Microstrain_" << i;
    micro_dof_names[i][0] = dofname.str();
  }
Teuchos::ArrayRCP<std::string> macro_dof_names_dotdot(numMacroScales);
Teuchos::ArrayRCP<std::string> macro_resid_names(numMacroScales);
Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_dof_names_dotdot(numMicroScales);
Teuchos::ArrayRCP< Teuchos::ArrayRCP<std::string> > micro_resid_names(numMicroScales);
if (supportsTransient){
  macro_dof_names_dotdot[0] = macro_dof_names[0]+"_dotdot";
  macro_resid_names[0] = macro_dof_names[0]+" Residual";
  for(int i=0;i<numMicroScales;i++){
    micro_dof_names_dotdot[i].resize(1);
    micro_resid_names[i].resize(1);
    micro_dof_names_dotdot[i][0] = micro_dof_names[i][0]+"_dotdot";
    micro_resid_names[i][0] = micro_dof_names[i][0]+" Residual";
  }
}
```

```
// 1.1 Gather Solution (displacement and acceleration)
```

New evaluator: Gather solution data from solver data structures to grid based structures. Note that accelerations are added as an evaluated field if appropriate.

Dependent Fields:

None.

Evaluated Fields:

u_{iI}	Nodal displacements	("Variable Name", "Displacement")	dims(cell,nNodes,vecDim)
a_{iI}	Nodal accelerations	("Variable Name", "Displacement_dotdot")	dims(cell,nNodes,vecDim)

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_GatherSolution_Def.hpp

```
   if (supportsTransient) fm0.template registerEvaluator<EvalT>
       (evalUtils.constructGatherSolutionEvaluator_withAcceleration(true, macro_dof_names, Teuchos::null, macro_dof_names_dotdot));
   else fm0.template registerEvaluator<EvalT>
       (evalUtils.constructGatherSolutionEvaluator_noTransient(true, macro_dof_names));

   int dof_offset = numDim; // dof layout is {x, y, ..., xx, xy, xz, yx, ...}
   int dof_stride = numDim*numDim;

// 1.1 Gather Solution (microstrains and micro accelerations)
```

New evaluator: Gather solution data from solver data structures to grid based structures. Note that micro accelerations are added as an evaluated field if appropriate.

Dependent Fields:

None.

Evaluated Fields:

ϵ^n_{ijI}	Nodal microstrains at scale 'n'	("Solution Name", "Microstrain_1")	dims(cell,nNodes,vecDim,vecDim)
$\ddot{\epsilon}^n_{iI}$	Nodal micro accelerations at scale 'n'	("Solution Name", "Microstrain_1_dotdot")	dims(cell,nNodes,vecDim,vecDim)

For implementation see:
problems/HMC_EvaluatorUtils_Def.hpp
evaluators/PHAL_GatherSolution_Def.hpp

```
   for(int i=0;i<numMicroScales;i++){
     if (supportsTransient) fm0.template registerEvaluator<EvalT>
       (evalUtilsHMC.constructGatherSolutionEvaluator_withAcceleration(
          micro_dof_names[i],
          Teuchos::null,
          micro_dof_names_dotdot[i],
          dof_offset+i*dof_stride));
     else fm0.template registerEvaluator<EvalT>
       (evalUtilsHMC.constructGatherSolutionEvaluator_noTransient(
          micro_dof_names[i],
          dof_offset+i*dof_stride));
   }

// 1.2  Gather Coordinates
```

New evaluator: Gather coordinate data from solver data structures to grid based structures. **Dependent Fields:** None.

Evaluated Fields:

x_{iI}	Nodal coordinates	(”Coordinate Vector Name”, ”Coord Vec”)	dims(cell,nNodes,vecDim)

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_GatherCoordinateVector_Def.hpp

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

// 2.1  Compute gradient matrix and weighted basis function values in current coordinates
```

Register new evaluator.
Dependent Fields:

x_{iI}	Nodal coordinates	("Cordinate Vector Name", "Coord Vec")	dims(cell,nNodes,vecDim)

Evaluated Fields:

$det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted measure	("Weights Name", "Weights")	dims(cell,nQPs)
$det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)$	Jacobian determinant	("Jacobian Det Name", Jacobian Det")	dims(cell,nQPs)
$N_I(\mathbf{x}_p)$	Basis function values	("BF Name", "BF")	dims(cell,nNodes,nQPs)
$N_I(\mathbf{x}_p)\ det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted ...	("Weighted BF Name", "wBF")	dims(cell,nNode,nQPs)
$\frac{\partial N_I(x_p)}{\partial \xi_k}J^{-1}_{kj}$	Gradient matrix wrt physical frame	("Gradient BF Name", "Gradient BF")	dims(cell,nNodes,nQPs,spcDim)
$\frac{\partial N_I(x_p)}{\partial \xi_k}J^{-1}_{kj}det\left(\frac{\partial x_{ip}}{\partial \xi_j}\right)\omega_p$	Weighted ...	("Weighted Gradient BF Name", "Weighted Gradient BF")	dims(cell,nNodes,nQPs,spcDim)

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_ComputeBasisFunctions_Def.hpp

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

// 3.1  Project displacements to Gauss points
```

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	("Variable Name", "Displacements")	dims(cell,nNodes,vecDim)
$N_I(\xi_p)$	Basis Functions	("BF Name", "BF")	dims(cell,nNodes,nQPs)

Evaluated Fields:

$u_i(\xi_p)$	Displacements at quadrature points	("Variable Name", "Displacements")	dims(cell,nQPs,vecDim)

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_DOFVecInterpolation_Def.hpp

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

// 3.1  Project microstrains to Gauss points
```

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	("Variable Name", "Displacements")	dims(cell,nNodes,vecDim)
$N_I(\xi_p)$	Basis Functions	("BF Name", "BF")	dims(cell,nNodes,nQPs)

Evaluated Fields:

$u_i(\xi_p)$	Displacements at quadrature points	("Variable Name", "Displacements")	dims(cell,nQPs,vecDim)

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_DOFVecInterpolation_Def.hpp

```
   for(int i=0;i<numMicroScales;i++)
     fm0.template registerEvaluator<EvalT>
       (evalUtilsHMC.constructDOFTensorInterpolationEvaluator(micro_dof_names[i][0]));

// 3.2  Project accelerations to Gauss points
```

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	("Variable Name", "Displacement_dotdot")	dims(cell,nNodes,vecDim)
$N_I(\xi_p)$	Basis Functions	("BF Name", "BF")	dims(cell,nNodes,nQPs)

Evaluated Fields:

$a_i(\xi_p)$	Acceleration at quadrature points	("Variable Name", "Dsplacement_dotdot")	dims(cell,nQPs,vecDim)

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_DOFVecInterpolation_Def.hpp

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

// 3.2  Project micro accelerations to Gauss points
```

New evaluator:

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

Dependent Fields:

$\ddot{\epsilon}^n_{ijI}$	Nodal micro acceleration	("Variable Name", "Microstrain_1_dotdot")	dims(cell,nNodes,vecDim,vecDim)
$N_I(\xi_p)$	Basis Functions	("BF Name", "BF")	dims(cell,nNodes,nQPs)

Evaluated Fields:

$\ddot{\epsilon}^n_{ij}(\xi_p)$	Micro acceleration at quadrature points	("Variable Name", "Microstrain_1_dotdot")	dims(cell,nQPs,vecDim,vecDim)

For implementation see:
HMC/problems/HMC_EvaluatorUtils_Def.hpp
HMC/evaluators/PHAL_DOFTensorInterpolation_Def.hpp

```
   if(supportsTransient)
     for(int i=0;i<numMicroScales;i++)
       fm0.template registerEvaluator<EvalT>
         (evalUtils.constructDOFVecInterpolationEvaluator(micro_dof_names_dotdot[i][0]));

// 3.3  Project nodal coordinates to Gauss points
```

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	("Coordinate Vector Name", "Coord Vec")	dims(cell,nNodes,vecDim)

Evaluated Fields:

x_{pi}	Gauss point coordinates	("Coordinate Vector Name", "Coord Vec")	dims(cell,nQPs,vecDim)

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_MapToPhysicalFrame_Def.hpp

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

// 3.4  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	("Variable Name", "Displacement")	dims(cell,nNodes,vecDim)
$B_I(\xi_p)$	Gradient of Basis Functions	("Gradient BF Name", "Grad BF")	dims(cell,nNodes,nQPs,vecDim)

Evaluated Fields:

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

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_DOFVecGradInterpolation_Def.hpp

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

// 3.5  Compute microstrain gradient
```

New evaluator:

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

Dependent Fields:

ϵ^n_{ijI}	Nodal microstrain at scale 'n'	("Variable Name", "Microstrain_1")	dims(cell,nNodes,vecDim,vecDim)
$B_I(\xi_p)$	Gradient of Basis Functions	("Gradient BF Name", "Grad BF")	dims(cell,nNodes,nQPs,vecDim)

Evaluated Fields:

$\left.\frac{\partial \epsilon^n_{ij}}{\partial x_k}\right\|_{\xi_p}$	Microstrain gradient	("Gradient Variable Name", "DOFTensorGrad Interpolation Microstrain_1")	dims(cell,nQPs,vecDim,vecDim,spcDim)

For implementation see:
HMC/problems/HMC_EvaluatorUtils_Def.hpp
HMC/evaluators/PHAL_DOFTensorGradInterpolation_Def.hpp

```
   for(int i=0;i<numMicroScales;i++)
     fm0.template registerEvaluator<EvalT>
       (evalUtilsHMC.constructDOFTensorGradInterpolationEvaluator(micro_dof_names[i][0]));

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

// 4.1  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 node vector	("Gradient Variable Name", "Displacement Gradient")	dims(cell,nQPs,vecDim,spcDim)

Evaluated Fields:

ϵ^p_{ij}	Infinitesimal strain	("Strain Name", "Strain")	dims(cell,nQPs,vecDim,spcDim)

For implementation see:
LCM/evaluators/Strain_Def.hpp

```
  {
    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);
  }

// 4.1  Compute microstrain difference
```

New evaluator:

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

Dependent Fields:

ϵ_{ij}^{p} Macro strain (”Gradient Variable Name”, ”Displacement Gradient”) dims(cell,nQPs,vecDim,spcDim)

Evaluated Fields:

ϵ_{ij}^{p} Infinitesimal strain (”Strain Name”, ”Strain”) dims(cell,nQPs,vecDim,spcDim)

For implementation see:
evaluators/HMC_StrainDifference_Def.hpp

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

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

// 5.1 Compute stresses
```

New evaluator:

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

Dependent Fields:

ϵ^p_{ij}	Infinitesimal strain	(”Strain Name”, ”Strain”)	dims(cell,nQPs,vecDim,spcDim)
ϵ^p_{ij}	Macro strain	(”Gradient Variable Name”, ”Displacement Gradient”)	dims(cell,nQPs,vecDim,spcDim)
$\left.\frac{\partial \epsilon^n_{ij}}{\partial x_k}\right\vert_{\xi_p}$	Microstrain gradient	(”Gradient Variable Name”, ”DOFTensorGrad Interpolation Microstrain_1”)	dims(cell,nQPs,vecDim,vecDim,spcDim)

Evaluated Fields:

σ^p_{ij}	Stress	(”Stress Name”, ”Stress”)	dims(cell,nQPs,vecDim,spcDim)

For implementation see:
LCM/evaluators/Stress_Def.hpp

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

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

```
//Input
//  Macro strain
p->set<std::string>("Strain Name", "Strain");
p->set< RCP<DataLayout> >("QP 2Tensor Data Layout", dl->qp_tensor);

//  Micro strains and micro strain gradients
for(int i=0;i<numMicroScales;i++){
  std::stringstream sdname; sdname << "Strain Difference " << i;
  std::string sd(sdname.str());
  sdname << " Name";
  p->set<std::string>(sdname.str(), sd);
  std::stringstream sdgradname; sdgradname << "Micro Strain Gradient " << i;
  std::string sdgrad(sdgradname.str());
  sdgradname << " Name";
  p->set<std::string>(sdgradname.str(), sdgrad);
}

//Output
p->set<std::string>("Stress Name", "Stress"); //dl->qp_tensor also
//
//  Micro stresses
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);

  std::string ds = Albany::strint("Double Stress",i);
  std::string dsname(ds); dsname += " Name";
  p->set<std::string>(dsname, ds);
}

//Parse material model constants
parseMaterialModel(p,params);

ev = rcp(new HMC::Stresses<EvalT,AlbanyTraits>(*p));
```

```
      fm0.template registerEvaluator<EvalT>(ev);
      p = stateMgr.registerStateVariable("Stress",dl->qp_tensor, dl->dummy, elementBlockName, "scalar", 0.0);
      ev = rcp(new PHAL::SaveStateField<EvalT,AlbanyTraits>(*p));
      fm0.template registerEvaluator<EvalT>(ev);
  }

// 6.1 Compute residual (stress divegence + inertia term)
```

New evaluator:

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

Dependent Fields:

σ^p_{ij}	Stress	("Stress Name", "Stress")	dims(cell,nQPs,vecDim,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 GradBF	("Weighted Gradient BF Name", "Weighted Gradient BF")	dims(cell,nNodes,nQPs,spcDim)
a^p_i	Acceleration at quadrature points	("Variable Name", "Dsplacement_dotdot")	dims(cell,nQPs,vecDim)
$N_I(\mathbf{x}_p) \ det \left(\frac{\partial x_{ip}}{\partial \xi_j} \right) \omega_p$	Weighted BF	("Weighted BF Name", "wBF")	dims(cell,nNode,nQPs)

Evaluated Fields:

$f_{iI}(x_i I)$	Residual	("Residual Name", "Residual")	dims(cell,nNodes,spcDim)

For implementation see:
LCM/evaluators/ElasticityResid_Def.hpp

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

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

```
    // \todo Is the required?
    p->set<std::string>("DefGrad Name", "Deformation Gradient"); //dl->qp_tensor also

    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", "Displacement_dotdot");
    p->set< RCP<DataLayout> >("QP Vector Data Layout", dl->qp_vector);

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

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

// X.X  Scatter nodal forces
```

New evaluator: Scatter the nodal forces (i.e., "Displacement Residual") from the grid based structures to the solver data structures. **Dependent Fields:**

u_{iI} Displacement residual ("Residual Name", "Displacement Residual") dims(cell,nNodes,vecDim)

Evaluated Fields:
None.

For implementation see:
problems/Albany_EvaluatorUtils_Def.hpp
evaluators/PHAL_ScatterResidual_Def.hpp

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

```
  { // 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, elementBlockName, "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);
    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
```