Examples

Example: 1

As Simple As It Gets

Listing 1: Example 1: ex01.i

```
[Mesh]
  file = mug.e
[]

[Variables]
  active = 'diffused'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff'

  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  active = 'bottom top'

  [./bottom]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom'
    value = 1
  [../]

  [./top]
    type = DirichletBC
    variable = diffused
    boundary = 'top'
    value = 0
  [../]
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 2: Example 1: main.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

/**
 * Example 1: Input File - The smallest MOOSE based application possible.  It solves
 * a simple 2D diffusion problem with Dirichlet boundary conditions using built-in
 * objects from MOOSE.
 */

#include "ExampleApp.h"
//Moose Includes
#include "MooseInit.h"
#include "Moose.h"
#include "MooseApp.h"
#include "AppFactory.h"

// Create a performance log
PerfLog Moose::perf_log("Example");

// Begin the main program.
int main(int argc, char *argv[])
{
  // Initialize MPI, solvers and MOOSE
  MooseInit init(argc, argv);

  // Register this application's MooseApp and any it depends on
  ExampleApp::registerApps();

  // This creates dynamic memory that we're responsible for deleting
  MooseApp * app = AppFactory::createApp("ExampleApp", argc, argv);

  // Execute the application
  app->run();

  // Free up the memory we created earlier
  delete app;

  return 0;
}
```

Listing 3: Example 1: ExampleApp.h

```
#ifndef EXAMPLEAPP_H
#define EXAMPLEAPP_H

#include "MooseApp.h"

class ExampleApp;

template<>
InputParameters validParams<ExampleApp>();

class ExampleApp : public MooseApp
{
public:
  ExampleApp(const std::string & name, InputParameters parameters);
  virtual ~ExampleApp();

  static void registerApps();
  static void registerObjects(Factory & factory);
  static void associateSyntax(Syntax& syntax, ActionFactory & action_factory);
};

#endif /* EXAMPLEAPP_H */
```

Listing 4: Example 1: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & /*factory*/)
{
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Listing 5: Example 1: Diffusion.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef DIFFUSION_H
#define DIFFUSION_H

#include "Kernel.h"

class Diffusion;

template<>
InputParameters validParams<Diffusion>();


class Diffusion : public Kernel
{
public:
  Diffusion(const std::string & name, InputParameters parameters);
  virtual ~Diffusion();

protected:
  virtual Real computeQpResidual();
  virtual Real computeQpJacobian();
};


#endif /* DIFFUSION_H */
```

Listing 6: Example 1: Diffusion.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "Diffusion.h"


template<>
InputParameters validParams<Diffusion>()
{
  InputParameters p = validParams<Kernel>();
  return p;
}


Diffusion::Diffusion(const std::string & name, InputParameters parameters) :
    Kernel(name, parameters)
{

}

Diffusion::~Diffusion()
{

}

Real
Diffusion::computeQpResidual()
{
  return _grad_u[_qp] * _grad_test[_i][_qp];
}

Real
Diffusion::computeQpJacobian()
{
  return _grad_phi[_j][_qp] * _grad_test[_i][_qp];
}
```

Example: 2

Adding a Custom Kernel

Listing 7: Example 2: ex02.i

```
[Mesh]
  file = mug.e
[]

[Variables]
  active = 'convected'

  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff conv'

  [./diff]
    type = Diffusion
    variable = convected
  [../]

  [./conv]
    type = Convection
    variable = convected
    velocity = '0.0 0.0 1.0'
  [../]
[]

[BCs]
  active = 'bottom top'

  [./bottom]
    type = DirichletBC
    variable = convected
    boundary = 'bottom'
    value = 1
  [../]

  [./top]
    type = DirichletBC
    variable = convected
    boundary = 'top'
    value = 0
  [../]
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 8: Example 2: Convection.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef CONVECTION_H
#define CONVECTION_H

#include "Kernel.h"

/**
 * The forward declaration is so that we can declare the validParams() function
 * before we actually define the class... that way the definition isn't lost
 * at the bottom of the file.
 */

// Forward Declarations
class Convection;

/**
 * validParams returns the parameters that this Kernel accepts / needs
 * The actual body of the function MUST be in the .C file.
 */
template<>
InputParameters validParams<Convection>();

/**
 * Define the Kernel for a convection operator that looks like:
 *
 * (V . grad(u), test)
 *
 * where V is a given constant velocity field.
 */
class Convection : public Kernel
{
public:

  /**
   * This is the constructor declaration.  This class takes a
   * string and a InputParameters object, just like other
   * Kernel-derived classes.
   */
  Convection(const std::string & name,
             InputParameters parameters);

protected:
  /**
   * Responsible for computing the residual at one quadrature point.
   * This function should always be defined in the .C file.
   */
  virtual Real computeQpResidual();

  /**
   * Responsible for computing the diagonal block of the preconditioning matrix.
   * This is essentially the partial derivative of the residual with respect to
   * the variable this kernel operates on ("u").
   *
   * Note that this can be an approximation or linearization.  In this case it's
   * not because the Jacobian of this operator is easy to calculate.
   *
```

```
   * This function should always be defined in the .C file.
   */
  virtual Real computeQpJacobian();

private:
  /**
   * A vector object for storing the velocity.  Convenient for
   * computing dot products.
   */
  RealVectorValue _velocity;
};

#endif // CONVECTION_H
```

Listing 9: Example 2: Convection.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "Convection.h"

/**
 * This function defines the valid parameters for
 * this Kernel and their default values
 */
template<>
InputParameters validParams<Convection>()
{
  InputParameters params = validParams<Kernel>();
  params.addRequiredParam<RealVectorValue>("velocity", "Velocity Vector");
  return params;
}

Convection::Convection(const std::string & name,
                       InputParameters parameters) :
  // You must call the constructor of the base class first
  Kernel(name, parameters),
   _velocity(getParam<RealVectorValue>("velocity"))
{}

Real Convection::computeQpResidual()
{
  // velocity * _grad_u[_qp] is actually doing a dot product
  return _test[_i][_qp]*(_velocity*_grad_u[_qp]);
}

Real Convection::computeQpJacobian()
{
  // the partial derivative of _grad_u is just _grad_phi[_j]
  return _test[_i][_qp]*(_velocity*_grad_phi[_j][_qp]);
}
```

Listing 10: Example 2: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

// Example 2 Includes
#include "Convection.h"           // <- New include for our custom kernel

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  // Register any custom objects you have built on the MOOSE Framework
  registerKernel(Convection);  // <- registration
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Listing 11: Example 2: ex02_oversample.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 2
  ny = 2
  nz = 0
  zmax = 0
  elem_type = QUAD9
[]

[Variables]
  [./diffused]
    order = SECOND
  [../]
[]

[Kernels]
  active = 'diff'
  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[DiracKernels]
  [./foo]
    variable = diffused
    type = ConstantPointSource
    value = 1
    point = '0.3 0.3 0.0'
  [../]
[]

[BCs]
  active = 'all'
  [./all]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom left right top'
    value = 0.0
  [../]
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'

[]

[Outputs]
  file_base = out_os
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
  [./oversample_2]
    type = Exodus
    file_base = oversample_2
    oversample = true
    refinements = 2
  [../]
  [./oversample_4]
   type = Exodus
   file_base = oversample_4
   oversample = true
```

```
   refinements = 4
 [../]
[]
```

Example: 3

Multiphysics Coupling

Listing 12: Example 3: ex03.i

```
[Mesh]
  file = mug.e
[]

[Variables]
  active = 'convected diffused'

  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff_convected conv diff_diffused'

  [./diff_convected]
    type = Diffusion
    variable = convected
  [../]

  [./conv]
    type = Convection
    variable = convected

    # Couple a variable into the convection kernel using local_name = simulationg_name syntax
    some_variable = diffused
  [../]

  [./diff_diffused]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  active = 'bottom_convected top_convected bottom_diffused top_diffused'

  [./bottom_convected]
    type = DirichletBC
    variable = convected
    boundary = 'bottom'
    value = 1
  [../]

  [./top_convected]
    type = DirichletBC
    variable = convected
    boundary = 'top'
    value = 0
  [../]

  [./bottom_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom'
    value = 2
  [../]

  [./top_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'top'
    value = 0
```

```
  [../]
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 13: Example 3: Convection.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef CONVECTION_H
#define CONVECTION_H

#include "Kernel.h"

class Convection;

template<>
InputParameters validParams<Convection>();

class Convection : public Kernel
{
public:

  Convection(const std::string & name,
             InputParameters parameters);

protected:

  virtual Real computeQpResidual();

  virtual Real computeQpJacobian();

private:

  VariableGradient & _grad_some_variable;
};

#endif //CONVECTION_H
```

Listing 14: Example 3: Convection.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "Convection.h"

template<>
InputParameters validParams<Convection>()
{
  InputParameters params = validParams<Kernel>();

  params.addRequiredCoupledVar("some_variable", "The gradient of this variable will be used as the
      velocity vector.");
  return params;
}

Convection::Convection(const std::string & name,
                       InputParameters parameters) :
    Kernel(name, parameters),
    _grad_some_variable(coupledGradient("some_variable"))
{}

Real Convection::computeQpResidual()
{
  return _test[_i][_qp]*(_grad_some_variable[_qp]*_grad_u[_qp]);
}

Real Convection::computeQpJacobian()
{
  return _test[_i][_qp]*(_grad_some_variable[_qp]*_grad_phi[_j][_qp]);
}
```

Listing 15: Example 3: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "AppFactory.h"

// Example 3 Includes
#include "Convection.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerKernel(Convection);
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 4

Custom Boundary Conditions

Listing 16: Example 4: dirichlet_bc.i

```
[Mesh]
  file = square.e
  uniform_refine = 4
[]

[Variables]
  active = 'convected diffused'

  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff_convected conv diff_diffused'

  [./diff_convected]
    type = Diffusion
    variable = convected
  [../]

  [./conv]
    type = Convection
    variable = convected
    some_variable = diffused
  [../]

  [./diff_diffused]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  active = 'left_convected right_convected_dirichlet left_diffused right_diffused'

  [./left_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left'
    value = 0
  [../]

  [./right_convected_dirichlet]
    type = CoupledDirichletBC
    variable = convected
    boundary = 'right'
    alpha = 2

    some_var = diffused
  [../]

  # Note: This BC is not active in this input file
  [./right_convected_neumann]
    type = CoupledNeumannBC
    variable = convected
    boundary = 'right'
    alpha = 2

    some_var = diffused
  [../]

  [./left_diffused]
```

```
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 1
  [../]

[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre    boomeramg'
[]

[Outputs]
  file_base = out_coupled_dirichlet
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 17: Example 4: CoupledDirichletBC.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef COUPLEDDIRICHLETBC_H
#define COUPLEDDIRICHLETBC_H

#include "NodalBC.h"

//Forward Declarations
class CoupledDirichletBC;

template<>
InputParameters validParams<CoupledDirichletBC>();

/**
 * Implements a coupled Dirichlet BC where u = alpha * some_var on the boundary.
 */
class CoupledDirichletBC : public NodalBC
{
public:

  /**
   * Factory constructor, takes parameters so that all derived classes can be built using the same
   * constructor.
   */
  CoupledDirichletBC(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeQpResidual();

private:
  /**
   * Multiplier on the boundary.
   */
  Real _alpha;

  /**
   * Holds the values at the quadrature points
   * of a coupled variable.
   */
  VariableValue & _some_var_val;
};

#endif //COUPLEDDIRICHLETBC_H
```

Listing 18: Example 4: CoupledDirichletBC.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "CoupledDirichletBC.h"

template<>
InputParameters validParams<CoupledDirichletBC>()
{
  InputParameters params = validParams<NodalBC>();

  // Here we are adding a parameter that will be extracted from the input file by the Parser
  params.addParam<Real>("alpha", 0.0, "Value multiplied by the coupled value on the boundary");
  params.addRequiredCoupledVar("some_var", "Value on the Boundary");
  return params;
}

CoupledDirichletBC::CoupledDirichletBC(const std::string & name, InputParameters parameters) :
    NodalBC(name, parameters),

    /**
     * Grab the parameter for the multiplier.
     */
    _alpha(getParam<Real>("alpha")),

    /**
     * Get a reference to the coupled variable's values.
     */
    _some_var_val(coupledValue("some_var"))
{}

Real
CoupledDirichletBC::computeQpResidual()
{
  return _u[_qp]-(_alpha*_some_var_val[_qp]);
}
```

Listing 19: Example 4: neumann_bc.i

```
[Mesh]
  file = square.e
  uniform_refine = 4
[]

[Variables]
  active = 'convected diffused'

  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff_convected conv diff_diffused'

  [./diff_convected]
    type = Diffusion
    variable = convected
  [../]

  [./conv]
    type = Convection
    variable = convected
    some_variable = diffused
  [../]

  [./diff_diffused]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  active = 'left_convected right_convected_neumann left_diffused right_diffused'

  [./left_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left'
    value = 0
  [../]

  # Note: This BC is not active in this input file
  [./right_convected_dirichlet]
    type = CoupledDirichletBC
    variable = convected
    boundary = 'right'
    alpha = 2

    some_var = diffused
  [../]

  [./right_convected_neumann]
    type = CoupledNeumannBC
    variable = convected
    boundary = 'right'
    alpha = 2

    some_var = diffused
  [../]

  [./left_diffused]
```

```
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 1
  [../]

[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre    boomeramg'
[]

[Outputs]
  file_base = out_coupled_neumann
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 20: Example 4: CoupledNeumannBC.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef COUPLEDNEUMANNBC_H
#define COUPLEDNEUMANNBC_H

#include "IntegratedBC.h"

//Forward Declarations
class CoupledNeumannBC;

template<>
InputParameters validParams<CoupledNeumannBC>();

/**
 * Implements a simple constant Neumann BC where grad(u)=alpha * v on the boundary.
 * Uses the term produced from integrating the diffusion operator by parts.
 */
class CoupledNeumannBC : public IntegratedBC
{
public:

  /**
   * Factory constructor, takes parameters so that all derived classes can be built using the same
   * constructor.
   */
  CoupledNeumannBC(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeQpResidual();

private:
  /**
   * Multiplier on the boundary.
   */
  Real _alpha;

  /**
   * Holds the values at the quadrature points
   * of a coupled variable.
   */
  VariableValue & _some_var_val;
};

#endif //COUPLEDNEUMANNBC_H
```

Listing 21: Example 4: CoupledNeumannBC.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "CoupledNeumannBC.h"

template<>
InputParameters validParams<CoupledNeumannBC>()
{
  InputParameters params = validParams<IntegratedBC>();

  // Here we are adding a parameter that will be extracted from the input file by the Parser
  params.addParam<Real>("alpha", 0.0, "Value multiplied by the coupled value on the boundary");
  params.addRequiredCoupledVar("some_var", "Flux Value at the Boundary");
  return params;
}

CoupledNeumannBC::CoupledNeumannBC(const std::string & name, InputParameters parameters) :
    IntegratedBC(name, parameters),
    _alpha(getParam<Real>("alpha")),
    _some_var_val(coupledValue("some_var"))
{}

Real
CoupledNeumannBC::computeQpResidual()
{
  return -_test[_i][_qp]*_alpha*_some_var_val[_qp];
}
```

Listing 22: Example 4: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

// Example 4 Includes
#include "Convection.h"
#include "GaussContForcing.h"
#include "CoupledDirichletBC.h"
#include "CoupledNeumannBC.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerKernel(Convection);
  registerKernel(GaussContForcing);                 // Extra forcing term
  registerBoundaryCondition(CoupledDirichletBC);    // Register our Boundary Conditions
  registerBoundaryCondition(CoupledNeumannBC);
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Listing 23: Example 4: periodic_bc.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 50
  ny = 50
  nz = 0

  xmax = 40
  ymax = 40
  zmax = 0
  elem_type = QUAD4
[]

[Variables]
  [./u]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  [./diff]
    type = Diffusion
    variable = u
  [../]

  [./forcing]
    type = GaussContForcing
    variable = u
    x_center = 2
    y_center = 4
  [../]

  [./dot]
    type = TimeDerivative
    variable = u
  [../]
[]

[BCs]
  [./Periodic]
    #Note: Enable either "auto" or both "manual" conditions for this example
    active = 'manual_x manual_y'

    # Can use auto_direction with Generated Meshes
    [./auto]
      variable = u
      auto_direction = 'x y'
    [../]

     # Use Translation vectors for everything else
     [./manual_x]
       variable = u
       primary = 'left'
       secondary = 'right'
       translation = '40 0 0'
     [../]

     [./manual_y]
       variable = u
       primary = 'bottom'
       secondary = 'top'
       translation = '0 40 0'
     [../]
  [../]
[]

[Executioner]
  type = Transient
```

```
  dt = 1
  num_steps = 20
[]

[Outputs]
  file_base = out_pbc
  interval = 1
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 24: Example 4: trapezoid.i

```
[Mesh]
  file = trapezoid.e
  uniform_refine = 1
[]

# Polar to Cartesian
# R = sqrt(x^2 + y^2)
# x = R * cos(theta)
# y = R * sin(theta)
[Functions]
  [./tr_x]
    type = ParsedFunction
    value = sqrt(x^2+y^2)*cos(2*pi/3)
  [../]

  [./tr_y]
    type = ParsedFunction
    value = sqrt(x^2+y^2)*sin(2*pi/3)
  [../]

  [./itr_x]
    type = ParsedFunction
    value = sqrt(x^2+y^2)*cos(0)
  [../]

  [./itr_y]
    type = ParsedFunction
    value = sqrt(x^2+y^2)*sin(0)  # Always Zero!
  [../]
[]

[Variables]
  [./u]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  [./diff]
    type = Diffusion
    variable = u
  [../]

  # A forcing term near the periodic boundary
  [./forcing]
    type = GaussContForcing
    variable = u
    x_center = 2
    y_center = -1
    x_spread = 0.25
    y_spread = 0.5
  [../]

  [./dot]
    type = TimeDerivative
    variable = u
  [../]
[]

[BCs]
  [./Periodic]
    [./x]
      primary = 1
      secondary = 4
      transform_func = 'tr_x tr_y'
      inv_transform_func = 'itr_x itr_y'
    [../]
  [../]
```

```
[]

[Executioner]
  type = Transient
  dt = 0.5
  num_steps = 6
[]

[Outputs]
  file_base = out_trapezoid
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Example: 5

Automatic Mesh Adaptivity

Listing 25: Example 5: ex05.i

```
[Mesh]
  file = cube-hole.e
[]

[Variables]
  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]
  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  [./example_diff]
    type = ExampleCoefDiffusion
    variable = convected
    coef = 0.125
  [../]
  [./conv]
    type = Convection
    variable = convected
    some_variable = diffused
  [../]
  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  # convected=0 on all vertical sides except the right (x-max)
  [./cylinder_convected]
    type = DirichletBC
    variable = convected
    boundary = inside
    value = 1
  [../]
  [./exterior_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left top bottom'
    value = 0
  [../]
  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = left
    value = 0
  [../]
  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = right
    value = 10
  [../]
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'

  l_tol = 1e-3
  nl_rel_tol = 1e-6
  nl_abs_tol = 1e-9
```

```
[]

[Adaptivity]
  marker = errorfrac
  steps = 2
  [./Indicators]
    [./error]
      type = GradientJumpIndicator
      variable = convected
    [../]
  [../]
  [./Markers]
    [./errorfrac]
      type = ErrorFractionMarker
      refine = 0.5
      coarsen = 0
      indicator = error
    [../]
  [../]
[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 26: Example 5: ExampleCoefDiffusion.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLECOEFDIFFUSION_H
#define EXAMPLECOEFDIFFUSION_H

#include "Kernel.h"

//Forward Declarations
class ExampleCoefDiffusion;

template<>
InputParameters validParams<ExampleCoefDiffusion>();

class ExampleCoefDiffusion : public Kernel
{
public:

  ExampleCoefDiffusion(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeQpResidual();

  virtual Real computeQpJacobian();

private:
  Real _coef;
};
#endif // EXAMPLECOEFDIFFUSION_H
```

Listing 27: Example 5: ExampleCoefDiffusion.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleCoefDiffusion.h"

template<>
InputParameters validParams<ExampleCoefDiffusion>()
{
  InputParameters params = validParams<Kernel>();
  params.set<Real>("coef")=0.0;
  return params;
}

ExampleCoefDiffusion::ExampleCoefDiffusion(const std::string & name, InputParameters parameters)
  :Kernel(name, parameters),
   _coef(getParam<Real>("coef"))
{}

Real
ExampleCoefDiffusion::computeQpResidual()
{
  return _coef*_grad_test[_i][_qp]*_grad_u[_qp];
}

Real
ExampleCoefDiffusion::computeQpJacobian()
{
  return _coef*_grad_test[_i][_qp]*_grad_phi[_j][_qp];
}
```

Example: 6

Transient Analysis

Listing 28: Example 6: ex06.i

```
[Mesh]
  file = cyl-tet.e
[]

[Variables]
  active = 'diffused'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff euler'

  [./diff]
    type = Diffusion
    variable = diffused
  [../]

  [./euler]
    type = ExampleTimeDerivative
    variable = diffused
    time_coefficient = 20.0
  [../]
[]

[BCs]
  active = 'bottom_diffused top_diffused'

  [./bottom_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom'
    value = 0
  [../]

  [./top_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'top'
    value = 1
  [../]

[]

[Executioner]
  type = Transient   # Here we use the Transient Executioner

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'



  num_steps = 75
  dt = 1
[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 29: Example 6: Diffusion.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef DIFFUSION_H
#define DIFFUSION_H

#include "Kernel.h"

class Diffusion;

template<>
InputParameters validParams<Diffusion>();


class Diffusion : public Kernel
{
public:
  Diffusion(const std::string & name, InputParameters parameters);
  virtual ~Diffusion();

protected:
  virtual Real computeQpResidual();
  virtual Real computeQpJacobian();
};


#endif /* DIFFUSION_H */
```

Listing 30: Example 6: Diffusion.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "Diffusion.h"


template<>
InputParameters validParams<Diffusion>()
{
  InputParameters p = validParams<Kernel>();
  return p;
}


Diffusion::Diffusion(const std::string & name, InputParameters parameters) :
    Kernel(name, parameters)
{

}

Diffusion::~Diffusion()
{

}

Real
Diffusion::computeQpResidual()
{
  return _grad_u[_qp] * _grad_test[_i][_qp];
}

Real
Diffusion::computeQpJacobian()
{
  return _grad_phi[_j][_qp] * _grad_test[_i][_qp];
}
```

Listing 31: Example 6: ExampleTimeDerivative.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLETIMEDERIVATIVE
#define EXAMPLETIMEDERIVATIVE

#include "TimeDerivative.h"

// Forward Declarations
class ExampleTimeDerivative;

template<>
InputParameters validParams<ExampleTimeDerivative>();

class ExampleTimeDerivative : public TimeDerivative
{
public:

  ExampleTimeDerivative(const std::string & name,
                        InputParameters parameters);

protected:
  virtual Real computeQpResidual();

  virtual Real computeQpJacobian();

  Real _time_coefficient;
};

#endif //EXAMPLETIMEDERIVATIVE
```

Listing 32: Example 6: ExampleTimeDerivative.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleTimeDerivative.h"

#include "Material.h"

template<>
InputParameters validParams<ExampleTimeDerivative>()
{
  InputParameters params = validParams<TimeDerivative>();
  params.addParam<Real>("time_coefficient", 1.0, "Time Coefficient");
  return params;
}

ExampleTimeDerivative::ExampleTimeDerivative(const std::string & name,
                                             InputParameters parameters) :
    TimeDerivative(name,parameters),
    // This kernel expects an input parameter named "time_coefficient"
    _time_coefficient(getParam<Real>("time_coefficient"))
{}

Real
ExampleTimeDerivative::computeQpResidual()
{
  return _time_coefficient*TimeDerivative::computeQpResidual();
}

Real
ExampleTimeDerivative::computeQpJacobian()
{
  return _time_coefficient*TimeDerivative::computeQpJacobian();
}
```

Listing 33: Example 6: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

// Example 6 Includes
#include "ExampleDiffusion.h"
#include "Convection.h"
#include "ExampleTimeDerivative.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerKernel(Convection);
  registerKernel(ExampleDiffusion);
  registerKernel(ExampleTimeDerivative);
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 7

Custom Initial Conditions

Listing 34: Example 7: transient.i

```
[Mesh]
  file = half-cone.e
[]

[Variables]
  active = 'diffused'

  [./diffused]
    # Note that we do not have the 'active' parameter here.  Since it
    # is missing we will automatically pickup all nested blocks
    order = FIRST
    family = LAGRANGE

    # Use the initial Condition block underneath the variable
    # for which we want to apply this initial condition
    [./InitialCondition]
      type = ExampleIC
      coefficient = 2.0;
    [../]
  [../]
[]

[Kernels]
  [./td]
    type = TimeDerivative
    variable = diffused
  [../]

  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  active = 'left right'

  [./left]
    type = DirichletBC
    variable = diffused
    boundary = 'top'
    value = 2
  [../]

  [./right]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom'
    value = 10
  [../]
[]

[Executioner]
  type = Transient
  dt = 0.1
  start_time = 0
  num_steps = 10

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  # Request that we output the initial condition so we can inspect
  # the values with our visualization tool
  output_initial = true
  exodus = true
```

```
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 35: Example 7: ExampleIC.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEIC_H
#define EXAMPLEIC_H

// MOOSE Includes
#include "InitialCondition.h"

// Forward Declarations
class ExampleIC;

template<>
InputParameters validParams<ExampleIC>();

/**
 * ExampleIC just returns a constant value.
 */
class ExampleIC : public InitialCondition
{
public:

  /**
   * Constructor: Same as the rest of the MOOSE Objects
   */
  ExampleIC(const std::string & name,
            InputParameters parameters);

  /**
   * The value of the variable at a point.
   *
   * This must be overriden by derived classes.
   */
  virtual Real value(const Point & p);

private:
  Real _coefficient;
};

#endif //EXAMPLEIC_H
```

Listing 36: Example 7: ExampleIC.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleIC.h"

template<>
InputParameters validParams<ExampleIC>()
{
  InputParameters params = validParams<InitialCondition>();
  params.addRequiredParam<Real>("coefficient", "The value of the initial condition");
  return params;
}

ExampleIC::ExampleIC(const std::string & name,
                     InputParameters parameters) :
    InitialCondition(name, parameters),
    _coefficient(getParam<Real>("coefficient"))
{}

Real
ExampleIC::value(const Point & p)
{
  /**
   * _value * x
   * The Point class is defined in libMesh.  The spatial
   * coordinates x,y,z can be accessed individually using
   * the parenthesis operator and a numeric index from 0..2
   */
  return 2.*_coefficient*std::abs(p(0));
}
```

Listing 37: Example 7: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

// Example 7 Includes
#include "ExampleIC.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  // Register our custom Initial Condition with the Factory
  registerInitialCondition(ExampleIC);
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 8

Material Properties

Listing 38: Example 8: ex08.i

```
[Mesh]
  file = reactor.e
  # Let's assign human friendly names to the blocks on the fly
  block_id = '1 2'
  block_name = 'fuel deflector'

  boundary_id = '4 5'
  boundary_name = 'bottom top'
[]

[Variables]
  [./diffused]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.5
  [../]

  [./convected]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.0
  [../]
[]

[Kernels]
  # This Kernel consumes a real-gradient material property from the active material
  [./convection]
    type = Convection
    variable = convected
  [../]

  [./diff_convected]
    type = Diffusion
    variable = convected
  [../]

  [./example_diff]
    # This Kernel uses "diffusivity" from the active material
    type = ExampleDiffusion
    variable = diffused
  [../]

  [./time_deriv_diffused]
    type = TimeDerivative
    variable = diffused
  [../]

  [./time_deriv_convected]
    type = TimeDerivative
    variable = convected
  [../]
[]

[BCs]
  [./bottom_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom'
    value = 0
  [../]

  [./top_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'top'
    value = 5
  [../]

  [./bottom_convected]
```

```
    type = DirichletBC
    variable = convected
    boundary = 'bottom'
    value = 0
  [../]

  [./top_convected]
    type = NeumannBC
    variable = convected
    boundary = 'top'
    value = 1
  [../]
[]

[Materials]
  [./example]
    type = ExampleMaterial
    block = 'fuel'
    diffusion_gradient = 'diffused'

    # Approximate Parabolic Diffusivity
    independent_vals = '0 0.25 0.5 0.75 1.0'
    dependent_vals = '1e-2 5e-3 1e-3 5e-3 1e-2'
  [../]

  [./example1]
    type = ExampleMaterial
    block = 'deflector'
    diffusion_gradient = 'diffused'

    # Constant Diffusivity
    independent_vals = '0 1.0'
    dependent_vals = '1e-1 1e-1'
  [../]
[]

[Executioner]
  type = Transient

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'

  dt = 0.1
  num_steps = 10
[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 39: Example 8: ExampleMaterial.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEMATERIAL_H
#define EXAMPLEMATERIAL_H

#include "Material.h"
#include "LinearInterpolation.h"

//Forward Declarations
class ExampleMaterial;

template<>
InputParameters validParams<ExampleMaterial>();

/**
 * Example material class that defines a few properties.
 */
class ExampleMaterial : public Material
{
public:
  ExampleMaterial(const std::string & name,
                  InputParameters parameters);

protected:
  virtual void computeQpProperties();

private:
  /**
   * This is the member reference that will hold the computed values
   * for the Real value property in this class.
   */
  MaterialProperty<Real> & _diffusivity;

  /**
   * Computed values for the Gradient value property in this class.
   */
  MaterialProperty<RealGradient> & _convection_velocity;

  /**
   * This is the member reference that will hold the gradient
   * of the coupled variable
   */
  VariableGradient & _diffusion_gradient;

  /**
   * This object returns a piecewise linear function based an a series
   * of points and their corresponding values
   */
  LinearInterpolation _piecewise_func;
};

#endif //EXAMPLEMATERIAL_H
```

Listing 40: Example 8: ExampleMaterial.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleMaterial.h"

template<>
InputParameters validParams<ExampleMaterial>()
{
  InputParameters params = validParams<Material>();

  // Vectors for Linear Interpolation
  params.addRequiredParam<std::vector<Real> >("independent_vals", "The vector of indepedent values for
      building the piecewise function");
  params.addRequiredParam<std::vector<Real> >("dependent_vals", "The vector of depedent values for
      building the piecewise function");

  params.addCoupledVar("diffusion_gradient", "The gradient of this variable will be used to compute a
      velocity vector property.");

  return params;
}

ExampleMaterial::ExampleMaterial(const std::string & name,
                                 InputParameters parameters) :
    Material(name, parameters),
    // Declare that this material is going to provide a Real
    // valued property named "diffusivity" that Kernels can use.
    _diffusivity(declareProperty<Real>("diffusivity")),

    // Declare that this material is going to provide a RealGradient
    // valued property named "convection_velocity" that Kernels can use.
    _convection_velocity(declareProperty<RealGradient>("convection_velocity")),

    // Get the reference to the variable coupled into this Material
    _diffusion_gradient(isCoupled("diffusion_gradient") ? coupledGradient("diffusion_gradient") :
        _grad_zero),

    _piecewise_func(getParam<std::vector<Real> >("independent_vals"),
                    getParam<std::vector<Real> >("dependent_vals"))
{}

void
ExampleMaterial::computeQpProperties()
{
  // We will compute the diffusivity based on the Linear Interpolation of the provided vectors in the z-
      direction
  _diffusivity[_qp] = _piecewise_func.sample(_q_point[_qp](2));

  _convection_velocity[_qp] = _diffusion_gradient[_qp];
}
```

Listing 41: Example 8: ExampleDiffusion.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEDIFFUSION_H
#define EXAMPLEDIFFUSION_H

#include "Diffusion.h"

//Forward Declarations
class ExampleDiffusion;

/**
 * validParams returns the parameters that this Kernel accepts / needs
 * The actual body of the function MUST be in the .C file.
 */
template<>
InputParameters validParams<ExampleDiffusion>();

class ExampleDiffusion : public Diffusion
{
public:

  ExampleDiffusion(const std::string & name,
                   InputParameters parameters);

protected:
  virtual Real computeQpResidual();
  virtual Real computeQpJacobian();

  /**
   * This MooseArray will hold the reference we need to our
   * material property from the Material class
   */
  MaterialProperty<Real> & _diffusivity;
};
#endif //EXAMPLEDIFFUSION_H
```

Listing 42: Example 8: ExampleDiffusion.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleDiffusion.h"

/**
 * This function defines the valid parameters for
 * this Kernel and their default values
 */
template<>
InputParameters validParams<ExampleDiffusion>()
{
  InputParameters params = validParams<Diffusion>();
  return params;
}


ExampleDiffusion::ExampleDiffusion(const std::string & name,
                                   InputParameters parameters) :
    Diffusion(name,parameters),
    _diffusivity(getMaterialProperty<Real>("diffusivity"))
{}

Real
ExampleDiffusion::computeQpResidual()
{
  // We're dereferencing the _diffusivity pointer to get to the
  // material properties vector... which gives us one property
  // value per quadrature point.

  // Also... we're reusing the Diffusion Kernel's residual
  // so that we don't have to recode that.
  return _diffusivity[_qp]*Diffusion::computeQpResidual();
}

Real
ExampleDiffusion::computeQpJacobian()
{
  // We're dereferencing the _diffusivity pointer to get to the
  // material properties vector... which gives us one property
  // value per quadrature point.

  // Also... we're reusing the Diffusion Kernel's residual
  // so that we don't have to recode that.
  return _diffusivity[_qp]*Diffusion::computeQpJacobian();
}
```

Listing 43: Example 8: Convection.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef CONVECTION_H
#define CONVECTION_H

#include "Kernel.h"

class Convection;

template<>
InputParameters validParams<Convection>();

class Convection : public Kernel
{
public:

  Convection(const std::string & name,
             InputParameters parameters);

protected:

  virtual Real computeQpResidual();

  virtual Real computeQpJacobian();

private:

  MaterialProperty<RealGradient> & _velocity;
};

#endif //CONVECTION_H
```

Listing 44: Example 8: Convection.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "Convection.h"

template<>
InputParameters validParams<Convection>()
{
  InputParameters params = validParams<Kernel>();
  return params;
}

Convection::Convection(const std::string & name,
                       InputParameters parameters) :
    Kernel(name, parameters),

    // Retrieve a gradient material property to use for the convection
    // velocity
    _velocity(getMaterialProperty<RealGradient>("convection_velocity"))
{}

Real Convection::computeQpResidual()
{
  return _test[_i][_qp]*(_velocity[_qp]*_grad_u[_qp]);
}

Real Convection::computeQpJacobian()
{
  return _test[_i][_qp]*(_velocity[_qp]*_grad_phi[_j][_qp]);
}
```

Listing 45: Example 8: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

// Example 8 Includes
#include "ExampleDiffusion.h"
#include "Convection.h"
#include "ExampleMaterial.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerKernel(Convection);

  // Our new Diffusion Kernel that accepts a material property
  registerKernel(ExampleDiffusion);

  // Register our new material class so we can use it.
  registerMaterial(ExampleMaterial);
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 9

Stateful Material Properties

Listing 46: Example 9: ex09.i

```
[Mesh]
  file = square.e
  uniform_refine = 4
[]

[Variables]
  active = 'convected diffused'

  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'convected_ie example_diff conv diffused_ie diff'

  [./convected_ie]
    type = TimeDerivative
    variable = convected
  [../]

  [./example_diff]
    # This Kernel uses "diffusivity" from the active material
    type = ExampleDiffusion
    variable = convected
  [../]

  [./conv]
    type = Convection
    variable = convected
    some_variable = diffused
  [../]

  [./diffused_ie]
    type = TimeDerivative
    variable = diffused
  [../]

  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  active = 'left_convected right_convected left_diffused right_diffused'

  [./left_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left'
    value = 0
  [../]

  [./right_convected]
    type = DirichletBC
    variable = convected
    boundary = 'right'
    value = 1

    some_var = diffused
  [../]
```

```
  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 1
  [../]

[]

[Materials]
  active = example_material

  [./example_material]
    type = ExampleMaterial
    block = 1
    initial_diffusivity = 0.05
  [../]
[]

[Executioner]
  type = Transient

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'



  num_steps = 10
  dt = 1.0
[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 47: Example 9: ExampleMaterial.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "Material.h"

#ifndef EXAMPLEMATERIAL_H
#define EXAMPLEMATERIAL_H

//Forward Declarations
class ExampleMaterial;

template<>
InputParameters validParams<ExampleMaterial>();

/**
 * Example material class that defines a few properties.
 */
class ExampleMaterial : public Material
{
public:
  ExampleMaterial(const std::string & name,
                  InputParameters parameters);

protected:
  virtual void initQpStatefulProperties();
  virtual void computeQpProperties();

private:
  Real _initial_diffusivity;

  /**
   * Create two MooseArray Refs to hold the current
   * and previous material properties respectively
   */
  MaterialProperty<Real> & _diffusivity;
  MaterialProperty<Real> & _diffusivity_old;
};

#endif //EXAMPLEMATERIAL_H
```

Listing 48: Example 9: ExampleMaterial.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleMaterial.h"

template<>
InputParameters validParams<ExampleMaterial>()
{
  InputParameters params = validParams<Material>();
  params.addParam<Real>("initial_diffusivity", 1.0, "The Initial Diffusivity");
  return params;
}

ExampleMaterial::ExampleMaterial(const std::string & name,
                                 InputParameters parameters) :
    Material(name, parameters),

    // Get a parameter value for the diffusivity
    _initial_diffusivity(getParam<Real>("initial_diffusivity")),

    // Declare that this material is going to have a Real
    // valued property named "diffusivity" that Kernels can use.
    _diffusivity(declareProperty<Real>("diffusivity")),

    // Declare that we are going to have an old value of diffusivity
    // Note: this is _expensive_ as we have to store values for each
    // qp throughout the mesh. Only do this if you REALLY need it!
    _diffusivity_old(declarePropertyOld<Real>("diffusivity"))
{}

void
ExampleMaterial::initQpStatefulProperties()
{
  // init the diffusivity property (this will become
  // _diffusivity_old in the first call of computeProperties)
  _diffusivity[_qp] = _initial_diffusivity;
}

void
ExampleMaterial::computeQpProperties()
{
  _diffusivity[_qp] = _diffusivity_old[_qp] * 2;
}
```

Listing 49: Example 9: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

// Example 9 Includes
#include "ExampleDiffusion.h"
#include "Convection.h"
#include "ExampleMaterial.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerKernel(Convection);
  registerKernel(ExampleDiffusion);
  registerMaterial(ExampleMaterial);
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 10

Auxiliary Variables

Listing 50: Example 10: ex10.i

```
[Mesh]
  file = car.e
[]

[Variables]
  active = 'diffused'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

# Here is the AuxVariables section where we declare the variables that
# will hold the AuxKernel calcuations.  The declaration syntax is very
# similar to that of the regular variables section
[AuxVariables]
  active = 'nodal_aux element_aux'

  [./nodal_aux]
    order = FIRST
    family = LAGRANGE
  [../]

  [./element_aux]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]

[Kernels]
  active = 'diff'

  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

# Here is the AuxKernels section where we enable the AuxKernels, link
# them to our AuxVariables, set coupling parameters, and set input parameters
[AuxKernels]
  active = 'nodal_example element_example'

  [./nodal_example]
    type = ExampleAux
    variable = nodal_aux
    value = 3.0
    coupled = diffused
  [../]

  [./element_example]
    type = ExampleAux
    variable = element_aux
    value = 4.0
    coupled = diffused
  [../]
[]

[BCs]
  active = 'bottom top'

  [./bottom]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom'
    value = 0
  [../]
```

```
  [./top]
    type = DirichletBC
    variable = diffused
    boundary = 'top'
    value = 1
  [../]
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 51: Example 10: ExampleAux.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEAUX_H
#define EXAMPLEAUX_H

#include "AuxKernel.h"


//Forward Declarations
class ExampleAux;

template<>
InputParameters validParams<ExampleAux>();

/**
 * Coupled auxiliary value
 */
class ExampleAux : public AuxKernel
{
public:

  /**
   * Factory constructor, takes parameters so that all derived classes can be built using the same
   * constructor.
   */
  ExampleAux(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeValue();

  VariableValue & _coupled_val;

  Real _value;
};

#endif //EXAMPLEAUX_H
```

Listing 52: Example 10: ExampleAux.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleAux.h"

template<>
InputParameters validParams<ExampleAux>()
{
  InputParameters params = validParams<AuxKernel>();
  params.addParam<Real>("value", 0.0, "Scalar value used for our auxiliary calculation");
  params.addRequiredCoupledVar("coupled", "Coupled variable");
  return params;
}

ExampleAux::ExampleAux(const std::string & name, InputParameters parameters) :
    AuxKernel(name, parameters),

    // We can couple in a value from one of our kernels with a call to coupledValueAux
    _coupled_val(coupledValue("coupled")),

    // Set our member scalar value from InputParameters (read from the input file)
    _value(getParam<Real>("value"))
{}

/**
 * Auxiliary Kernels override computeValue() instead of computeQpResidual().  Aux Variables
 * are calculated either one per elemenet or one per node depending on whether we declare
 * them as "Elemental (Constant Monomial)" or "Nodal (First Lagrange)".  No changes to the
 * source are necessary to switch from one type or the other.
 */
Real
ExampleAux::computeValue()
{
  return _coupled_val[_qp] + _value;
}
```

Listing 53: Example 10: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "Factory.h"
#include "AppFactory.h"

// Example 10 Includes
#include "ExampleAux.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerObjects(Factory & factory)
{
  // Register our Example AuxKernel with the AuxFactory
  registerAux(ExampleAux);
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::associateSyntax(Syntax& /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 11

Preconditioning

Listing 54: Example 11: default.i

```
[Mesh]
  file = square.e
[]

[Variables]
  active = 'diffused forced'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]

  [./forced]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff_diffused conv_forced diff_forced'

  [./diff_diffused]
    type = Diffusion
    variable = diffused
  [../]

  [./conv_forced]
    type = CoupledForce
    variable = forced
    v = diffused
  [../]

  [./diff_forced]
    type = Diffusion
    variable = forced
  [../]
[]

[BCs]
  active = 'left_diffused right_diffused left_forced'

  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 100
  [../]

  [./left_forced]
    type = DirichletBC
    variable = forced
    boundary = 'left'
    value = 0
  [../]

  [./right_forced]
    type = DirichletBC
    variable = forced
    boundary = 'right'
    value = 0
  [../]
[]
```

```
[Executioner]
  type = Steady

#  petsc_options = '-snes_mf'

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'



#  petsc_options_iname = '-pc_type'
#  petsc_options_value = 'lu'
[]

[Outputs]
  file_base = out
  output_initial = true
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 55: Example 11: smp.i

```
[Mesh]
  file = square.e
[]

[Variables]
  active = 'diffused forced'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]

  [./forced]
    order = FIRST
    family = LAGRANGE
  [../]
[]

# The Preconditioning block
[Preconditioning]
  active = 'SMP_jfnk'

  [./SMP_jfnk]
    type = SMP

    off_diag_row    = 'forced'
    off_diag_column = 'diffused'


  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


    petsc_options_iname = '-pc_type'
    petsc_options_value = 'lu'
  [../]

  [./SMP_jfnk_full]
    type = SMP

    full = true


  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


    petsc_options_iname = '-pc_type'
    petsc_options_value = 'lu'
  [../]

  [./SMP_n]
    type = SMP

    off_diag_row    = 'forced'
    off_diag_column = 'diffused'


    solve_type = 'NEWTON'

    petsc_options_iname = '-pc_type'
    petsc_options_value = 'lu'
  [../]
[]

[Kernels]
  active = 'diff_diffused conv_forced diff_forced'

  [./diff_diffused]
```

```
    type = Diffusion
    variable = diffused
  [../]

  [./conv_forced]
    type = CoupledForce
    variable = forced
    v = diffused
  [../]

  [./diff_forced]
    type = Diffusion
    variable = forced
  [../]
[]

[BCs]
  active = 'left_diffused right_diffused left_forced'

  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 1
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 2
    value = 100
  [../]

  [./left_forced]
    type = DirichletBC
    variable = forced
    boundary = 1
    value = 0
  [../]

  [./right_forced]
    type = DirichletBC
    variable = forced
    boundary = 2
    value = 0
  [../]
[]

[Executioner]
  type = Steady



[]

[Outputs]
  file_base = out
  output_initial = true
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 56: Example 11: fdp.i

```
[Mesh]
  file = square.e
[]

[Variables]
  active = 'diffused forced'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]

  [./forced]
    order = FIRST
    family = LAGRANGE
  [../]
[]

# The Preconditioning block
[Preconditioning]
  active = 'FDP_jfnk'

  [./FDP_jfnk]
    type = FDP

    off_diag_row    = 'forced'
    off_diag_column = 'diffused'


  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


    petsc_options_iname = '-pc_type -mat_fd_coloring_err -mat_fd_type'
    petsc_options_value = 'lu       1e-6                 ds'
  [../]

  [./FDP_n]
    type = FDP

    off_diag_row    = 'forced'
    off_diag_column = 'diffused'


    solve_type = 'NEWTON'

    petsc_options_iname = '-pc_type -mat_fd_coloring_err -mat_fd_type'
    petsc_options_value = 'lu       1e-6                 ds'
  [../]

  [./FDP_n_full]
    type = FDP

    full = true


    solve_type = 'NEWTON'

    petsc_options_iname = '-pc_type -mat_fd_coloring_err -mat_fd_type'
    petsc_options_value = 'lu       1e-6                 ds'
  [../]
[]

[Kernels]
  active = 'diff_diffused conv_forced diff_forced'

  [./diff_diffused]
    type = Diffusion
    variable = diffused
```

```
  [../]

  [./conv_forced]
    type = CoupledForce
    variable = forced
    v = diffused
  [../]

  [./diff_forced]
    type = Diffusion
    variable = forced
  [../]
[]

[BCs]
  active = 'left_diffused right_diffused left_forced'

  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 100
  [../]

  [./left_forced]
    type = DirichletBC
    variable = forced
    boundary = 'left'
    value = 0
  [../]

  [./right_forced]
    type = DirichletBC
    variable = forced
    boundary = 'right'
    value = 0
  [../]
[]

[Executioner]
  type = Steady



[]

[Outputs]
  file_base = out
  output_initial = true
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 57: Example 11: CoupledForce.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef COUPLEDFORCE_H
#define COUPLEDFORCE_H

#include "Kernel.h"

// Forward Declaration
class CoupledForce;

template<>
InputParameters validParams<CoupledForce>();

/**
 * Simple class to demonstrate off diagonal Jacobian contributions.
 */
class CoupledForce : public Kernel
{
public:
  CoupledForce(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeQpResidual();

  virtual Real computeQpJacobian();

  virtual Real computeQpOffDiagJacobian(unsigned int jvar);

private:
  unsigned int _v_var;
  VariableValue & _v;
};

#endif //COUPLEDFORCE_H
```

Listing 58: Example 11: CoupledForce.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "CoupledForce.h"

template<>
InputParameters validParams<CoupledForce>()
{
  InputParameters params = validParams<Kernel>();

  params.addRequiredCoupledVar("v", "The coupled variable which provides the force");

  return params;
}

CoupledForce::CoupledForce(const std::string & name, InputParameters parameters) :
    Kernel(name, parameters),
    _v_var(coupled("v")),
    _v(coupledValue("v"))
{
}

Real
CoupledForce::computeQpResidual()
{
  return -_v[_qp]*_test[_i][_qp];
}

Real
CoupledForce::computeQpJacobian()
{
  return 0;
}

Real
CoupledForce::computeQpOffDiagJacobian(unsigned int jvar)
{
  if (jvar == _v_var)
    return -_phi[_j][_qp]*_test[_i][_qp];
  return 0.0;
}
```

Example: 12

Physics Based Preconditioning

Listing 59: Example 12: ex12.i

```
[Mesh]
  file = square.e
[]

[Variables]
  active = 'diffused forced'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]

  [./forced]
    order = FIRST
    family = LAGRANGE
  [../]
[]

# The Preconditioning block
[Preconditioning]
  active = 'PBP'

  [./PBP]
    type = PBP
    solve_order = 'diffused forced'
    preconditioner  = 'LU LU'
    off_diag_row    = 'forced'
    off_diag_column = 'diffused'
  [../]
[]

[Kernels]
  active = 'diff_diffused conv_forced diff_forced'

  [./diff_diffused]
    type = Diffusion
    variable = diffused
  [../]

  [./conv_forced]
    type = CoupledForce
    variable = forced
    v = diffused
  [../]

  [./diff_forced]
    type = Diffusion
    variable = forced
  [../]
[]

[BCs]
  active = 'left_diffused right_diffused left_forced'

  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 100
  [../]

  [./left_forced]
```

```
    type = DirichletBC
    variable = forced
    boundary = 'left'
    value = 0
  [../]

  [./right_forced]
    type = DirichletBC
    variable = forced
    boundary = 'right'
    value = 0
  [../]
[]

[Executioner]
  type = Steady



[]

[Outputs]
  file_base = out
  output_initial = true
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 60: Example 12: CoupledForce.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef COUPLEDFORCE_H
#define COUPLEDFORCE_H

#include "Kernel.h"

// Forward Declaration
class CoupledForce;

template<>
InputParameters validParams<CoupledForce>();

/**
 * Simple class to demonstrate off diagonal Jacobian contributions.
 */
class CoupledForce : public Kernel
{
public:
  CoupledForce(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeQpResidual();

  virtual Real computeQpJacobian();

  virtual Real computeQpOffDiagJacobian(unsigned int jvar);

private:
  unsigned int _v_var;
  VariableValue & _v;
};

#endif //COUPLEDFORCE_H
```

Listing 61: Example 12: CoupledForce.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "CoupledForce.h"

template<>
InputParameters validParams<CoupledForce>()
{
  InputParameters params = validParams<Kernel>();

  params.addRequiredCoupledVar("v", "The coupled variable which provides the force");

  return params;
}

CoupledForce::CoupledForce(const std::string & name, InputParameters parameters) :
    Kernel(name, parameters),
    _v_var(coupled("v")),
    _v(coupledValue("v"))
{
}

Real
CoupledForce::computeQpResidual()
{
  return -_v[_qp]*_test[_i][_qp];
}

Real
CoupledForce::computeQpJacobian()
{
  return 0;
}

Real
CoupledForce::computeQpOffDiagJacobian(unsigned int jvar)
{
  if (jvar == _v_var)
    return -_phi[_j][_qp]*_test[_i][_qp];
  return 0.0;
}
```

Example: 13

Custom Functions

Listing 62: Example 13: ex13.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2

  nx = 100
  ny = 100

  xmin = 0.0
  xmax = 1.0

  ymin = 0.0
  ymax = 1.0
[]

[Variables]
  active = 'forced'

  [./forced]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Functions]
  active = 'bc_func forcing_func'

  # A ParsedFunction allows us to supply analytic expressions
  # directly in the input file
  [./bc_func]
    type = ParsedFunction
    value = sin(alpha*pi*x)
    vars = 'alpha'
    vals = '16'
  [../]

  # This function is an actual compiled function
  # We could have used ParsedFunction for this as well
  [./forcing_func]
    type = ExampleFunction
    alpha = 16
  [../]
[]

[Kernels]
  active = 'diff forcing'

  [./diff]
    type = Diffusion
    variable = forced
  [../]

  # This Kernel can take a function name to use
  [./forcing]
    type = UserForcingFunction
    variable = forced
    function = forcing_func
  [../]
[]

[BCs]
  active = 'all'

  # The BC can take a function name to use
  [./all]
    type = FunctionDirichletBC
    variable = forced
    boundary = 'bottom right top left'
    function = bc_func
  [../]
```

```
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 63: Example 13: ExampleFunction.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEFUNCTION_H
#define EXAMPLEFUNCTION_H

#include "Function.h"

class ExampleFunction;

template<>
InputParameters validParams<ExampleFunction>();

class ExampleFunction : public Function
{
public:
  ExampleFunction(const std::string & name, InputParameters parameters);

  virtual Real value(Real t, const Point & p);

protected:
  Real _alpha;
};

#endif //EXAMPLEFUNCTION_H
```

Listing 64: Example 13: ExampleFunction.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleFunction.h"

template<>
InputParameters validParams<ExampleFunction>()
{
  InputParameters params = validParams<Function>();
  params.addParam<Real>("alpha", 1.0, "The value of alpha");
  return params;
}

ExampleFunction::ExampleFunction(const std::string & name, InputParameters parameters) :
    Function(name, parameters),
    _alpha(getParam<Real>("alpha"))
{}

Real
ExampleFunction::value(Real /*t*/, const Point & p)
{
  return _alpha*_alpha*libMesh::pi*libMesh::pi*std::sin(_alpha*libMesh::pi*p(0));  // p(0) == x
}
```

Listing 65: Example 13: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "AppFactory.h"

// Example 13 Includes
#include "ExampleFunction.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerFunction(ExampleFunction);
}

void
ExampleApp::associateSyntax(Syntax & /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 14

Postprocessors and Code Verification

Listing 66: Example 14: ex14.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2

  nx = 10
  ny = 10

  xmin = 0.0
  xmax = 1.0

  ymin = 0.0
  ymax = 1.0
[]

[Variables]
  active = 'forced'

  [./forced]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Functions]
  active = 'bc_func forcing_func'

  # A ParsedFunction allows us to supply analytic expressions
  # directly in the input file
  [./bc_func]
    type = ParsedFunction
    value = sin(alpha*pi*x)
    vars = 'alpha'
    vals = '16'
  [../]

  # This function is an actual compiled function
  # We could have used ParsedFunction for this as well
  [./forcing_func]
    type = ExampleFunction
    alpha = 16
  [../]
[]

[Kernels]
  active = 'diff forcing'

  [./diff]
    type = Diffusion
    variable = forced
  [../]

  # This Kernel can take a function name to use
  [./forcing]
    type = UserForcingFunction
    variable = forced
    function = forcing_func
  [../]
[]

[BCs]
  active = 'all'

  # The BC can take a function name to use
  [./all]
    type = FunctionDirichletBC
    variable = forced
    boundary = 'bottom right top left'
    function = bc_func
  [../]
```

```
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Adaptivity]
  marker = uniform

  # Uniformly refine the mesh
  # for the convergence study
  [./Markers]
    type = UniformMarker
    mark = REFINE
  [../]
[]

[Postprocessors]
  [./dofs]
    type = NumDOFs
  [../]

  [./integral]
    type = ElementL2Error
    variable = forced
    function = bc_func
  [../]
[]

[Outputs]
  file_base = out
  exodus = true
  csv = true
  x[./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Example: 15

Custom Action

Listing 67: Example 15: ex15.i

```
[Mesh]
  file = square.e
  uniform_refine = 4
[]

[Variables]
  active = 'convected diffused'

  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

# This is our new custom Convection Diffusion "Meta" block
# that adds multiple kernels into our simulation
#
# Convection and Diffusion kernels on the first variable
# Diffusion kernel on the second variable
# The Convection kernel is coupled to the Diffusion kernel on the second variable
[ConvectionDiffusion]
    variables = 'convected diffused'
[]

[BCs]
  active = 'left_convected right_convected left_diffused right_diffused'

  [./left_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left'
    value = 0
  [../]

  [./right_convected]
    type = DirichletBC
    variable = convected
    boundary = 'right'
    value = 1

    some_var = diffused
  [../]

  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 1
  [../]

[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
```

```
[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 68: Example 15: ConvectionDiffusionAction.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef CONVECTIONDIFFUSIONACTION_H
#define CONVECTIONDIFFUSIONACTION_H

#include "Action.h"

class ConvectionDiffusionAction : public Action
{
public:
  ConvectionDiffusionAction(const std::string & name, InputParameters params);

  virtual void act();
};

template<>
InputParameters validParams<ConvectionDiffusionAction>();

#endif //CONVECTIONDIFFUSIONACTION_H
```

Listing 69: Example 15: ConvectionDiffusionAction.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ConvectionDiffusionAction.h"
#include "Factory.h"
#include "Parser.h"
#include "FEProblem.h"

template<>
InputParameters validParams<ConvectionDiffusionAction>()
{
  InputParameters params = validParams<Action>();
  params.addRequiredParam<std::vector<NonlinearVariableName> >("variables", "The names of the convection
       and diffusion variables in the simulation");

  return params;
}

ConvectionDiffusionAction::ConvectionDiffusionAction(const std::string & name, InputParameters params) :
    Action(name, params)
{
}

void
ConvectionDiffusionAction::act()
{
  std::vector<NonlinearVariableName> variables = getParam<std::vector<NonlinearVariableName> > ("
      variables");
  std::vector<VariableName> vel_vec_variable;

  /**
   * We need to manually setup our Convection-Diffusion and Diffusion variables on our two
   * variables we are expecting from the input file.  Much of the syntax below is hidden by the
   * parser system but we have to set things up ourselves this time.
   */

  // Do some error checking
  mooseAssert(variables.size() == 2, "Expected 2 variables, received " + variables.size());

  // Setup our Diffusion Kernel on the "u" variable
  {
    InputParameters params = _factory.getValidParams("Diffusion");
    params.set<NonlinearVariableName>("variable") = variables[0];
    _problem->addKernel("Diffusion", "diff_u", params);
  }

  // Setup our Convection Kernel on the "u" variable coupled to the diffusion variable "v"
  {
    InputParameters params = _factory.getValidParams("Convection");
    params.set<NonlinearVariableName>("variable") = variables[0];
//    params.addCoupledVar("some_variable", "The gradient of this var");
    vel_vec_variable.push_back(variables[1]);
    params.set<std::vector<VariableName> >("some_variable") = vel_vec_variable;
    _problem->addKernel("Convection", "conv", params);
  }

  // Setup out Diffusion Kernel on the "v" variable
  {
```

```
    InputParameters params = _factory.getValidParams("Diffusion");
    params.set<NonlinearVariableName>("variable") = variables[1];
    _problem->addKernel("Diffusion", "diff_v", params);
  }
}
```

Listing 70: Example 15: ExampleApp.C

```
#include "Moose.h"
#include "ExampleApp.h"
#include "AppFactory.h"
#include "ActionFactory.h"  // <- Actions are special (they have their own factory)
#include "Syntax.h"

// Example 15 Includes
#include "Convection.h"
#include "ConvectionDiffusionAction.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}


void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerKernel(Convection);
}

void
ExampleApp::associateSyntax(Syntax & syntax, ActionFactory & action_factory)
{
  /**
   * Registering an Action is a little different than registering the other MOOSE
   * objects.  First, you need to register your Action in the associateSyntax method.
   * Also, you register your Action class with an "action_name" that can be
   * satisfied by executing the Action (running the "act" virtual method).
   */
  registerAction(ConvectionDiffusionAction, "add_kernel");

  /**
   * We need to tell the parser what new section name to look for and what
   * Action object to build when it finds it.  This is done directly on the syntax
   * with the registerActionSyntax method.
   *
   * The section name should be the "full path" of the parsed section but should NOT
   * contain a leading slash.  Wildcard characters can be used to replace a piece of the
   * path.
   */
  syntax.registerActionSyntax("ConvectionDiffusionAction", "ConvectionDiffusion");
}
```

Example: 16

Creating a Custom TimeStepper

Listing 71: Example 16: ex16.i

```
[Mesh]
  file = square.e
  uniform_refine = 4
[]

[Variables]
  active = 'convected diffused'

  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'example_diff conv diff euler'

  [./example_diff]
    type = ExampleDiffusion
    variable = convected
  [../]

  [./conv]
    type = Convection
    variable = convected
    some_variable = diffused
  [../]

  [./diff]
    type = Diffusion
    variable = diffused
  [../]

  [./euler]
    type = ExampleImplicitEuler
    variable = diffused
  [../]
[]

[BCs]
  active = 'left_convected right_convected left_diffused right_diffused'

  [./left_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left'
    value = 0
  [../]

  [./right_convected]
    type = DirichletBC
    variable = convected
    boundary = 'right'
    value = 1
  [../]

  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
```

```
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 1
  [../]

[]

[Materials]
  active = 'example'

  [./example]
    type = ExampleMaterial
    block = 1
    diffusivity = 0.5
    time_coefficient = 20.0
  [../]
[]

[Executioner]
  type = Transient


  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'



  num_steps = 40

  # Use our custom TimeStepper
  [./TimeStepper]
    type = TransientHalf
    ratio = 0.5
    min_dt = 0.01
    dt = 1
  [../]
[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 72: Example 16: TransientHalf.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef TRANSIENTHALF_H
#define TRANSIENTHALF_H

#include "TimeStepper.h"

// Forward Declarations
class TransientHalf;

template<>
InputParameters validParams<TransientHalf>();

/**
 * This class cuts the timestep in half at every iteration
 * until it reaches a user-specified minimum value.
 */
class TransientHalf : public TimeStepper
{
public:

  TransientHalf(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeInitialDT();

  virtual Real computeDT();

private:
  Real _ratio;
  Real _min_dt;
};

#endif //TRANSIENTHALF_H
```

Listing 73: Example 16: TransientHalf.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "TransientHalf.h"

//Moose includes

template<>
InputParameters validParams<TransientHalf>()
{
  InputParameters params = validParams<TimeStepper>();
  params.addParam<Real>("dt", 1., "The initial time step size.");
  params.addParam<Real>("ratio", 0.5, "The ratio used to calculate the next timestep");
  params.addParam<Real>("min_dt", 0.01, "The smallest timestep we will allow");
  return params;
}

TransientHalf::TransientHalf(const std::string & name, InputParameters parameters) :
    TimeStepper(name, parameters),
    _ratio(getParam<Real>("ratio")),
    _min_dt(getParam<Real>("min_dt"))
{
}

Real
TransientHalf::computeInitialDT()
{
  return getParam<Real>("dt");
}

Real
TransientHalf::computeDT()
{
  /**
   * We won't grow timesteps with this example so if the ratio > 1.0 we'll just
   * leave current_dt alone.
   */
  if (_ratio < 1.0)
    // Shrink our timestep by the specified ratio or return the min if it's too small
    return std::max(getCurrentDT() * _ratio, _min_dt);
  else
    return getCurrentDT();
}
```

Listing 74: Example 16: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "AppFactory.h"

// Example 16 Includes
#include "TransientHalf.h"
#include "ExampleDiffusion.h"
#include "Convection.h"
#include "ExampleImplicitEuler.h"
#include "ExampleMaterial.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::registerObjects(Factory & factory)
{
  // Register our new executioner
  registerExecutioner(TransientHalf);
  registerKernel(ExampleDiffusion);
  registerKernel(Convection);
  registerKernel(ExampleImplicitEuler);
  registerMaterial(ExampleMaterial);
}

void
ExampleApp::associateSyntax(Syntax & /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 17

Adding a DiracKernel

Listing 75: Example 17: ex17.i

```
[Mesh]
  file = 3-4-torus.e
[]

[Variables]
  active = 'diffused'

  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]
[Kernels]
  active = 'diff'

  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[DiracKernels]
  active = 'example_point_source'

  [./example_point_source]
    type = ExampleDirac
    variable = diffused
    value = 1.0
    point = '-2.1 -5.08 0.7'
  [../]
[]

[BCs]
  active = 'left right'

  [./right]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 0
  [../]

  [./left]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 1
  [../]
[]

# The Preconditioning block
[Preconditioning]
  active = 'pbp'

  [./pbp]
    type = PBP
    solve_order = 'diffused'
    preconditioner  = 'AMG'
  [../]
[]

[Executioner]
  type = Steady

  solve_type = JFNK

[]

[Outputs]
```

```
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 76: Example 17: ExampleDirac.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEDIRAC_H
#define EXAMPLEDIRAC_H

// Moose Includes
#include "DiracKernel.h"

//Forward Declarations
class ExampleDirac;

template<>
InputParameters validParams<ExampleDirac>();

class ExampleDirac : public DiracKernel
{
public:
  ExampleDirac(const std::string & name, InputParameters parameters);

  virtual void addPoints();
  virtual Real computeQpResidual();

protected:
  Real _value;
  std::vector<Real> _point_param;
  Point _p;
};

#endif //EXAMPLEDIRAC_H
```

Listing 77: Example 17: ExampleDirac.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleDirac.h"

template<>
InputParameters validParams<ExampleDirac>()
{
  InputParameters params = validParams<DiracKernel>();
  params.addRequiredParam<Real>("value", "The value of the point source");
  params.addRequiredParam<std::vector<Real> >("point", "The x,y,z coordinates of the point");
  return params;
}

ExampleDirac::ExampleDirac(const std::string & name, InputParameters parameters) :
    DiracKernel(name, parameters),
    _value(getParam<Real>("value")),
    _point_param(getParam<std::vector<Real> >("point"))
{
  _p(0) = _point_param[0];

  if (_point_param.size() > 1)
  {
    _p(1) = _point_param[1];

    if (_point_param.size() > 2)
    {
      _p(2) = _point_param[2];
    }
  }
}

void
ExampleDirac::addPoints()
{
  // Add a point from the input file
  addPoint(_p);

  // Just add an arbitrary point
  addPoint(Point(4.9, 0.9, 0.9));
}

Real
ExampleDirac::computeQpResidual()
{
//  This is negative because it's a forcing function that has been brought over to the left side
  return -_test[_i][_qp]*_value;
}
```

Listing 78: Example 17: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"

#include "Moose.h"
#include "AppFactory.h"

#include "Convection.h"
#include "ExampleDirac.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerKernel(Convection);
  registerDiracKernel(ExampleDirac);  // <- registration
}

void
ExampleApp::associateSyntax(Syntax & /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 18

Coupling ODE into PDE

Listing 79: Example 18: ex18.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  xmin = 0
  xmax = 1
  ymin = 0
  ymax = 1
  nx = 10
  ny = 10
  elem_type = QUAD4
[]

[Functions]
  # ODEs
  [./exact_x_fn]
    type = ParsedFunction
    value = (-1/3)*exp(-t)+(4/3)*exp(5*t)
  [../]
[]

[Variables]
  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]

  # ODE variables
  [./x]
    family = SCALAR
    order = FIRST
    initial_condition = 1
  [../]
  [./y]
    family = SCALAR
    order = FIRST
    initial_condition = 2
  [../]

[]

[Kernels]
  [./td]
    type = TimeDerivative
    variable = diffused
  [../]
  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[ScalarKernels]
  [./td1]
    type = ODETimeDerivative
    variable = x
  [../]
  [./ode1]
    type = ImplicitODEx
    variable = x
    y = y
  [../]

  [./td2]
    type = ODETimeDerivative
    variable = y
  [../]
  [./ode2]
    type = ImplicitODEy
    variable = y
```

```
    x = x
  [../]
[]


[BCs]
  [./left]
    type = ScalarDirichletBC
    variable = diffused
    boundary = 1
    scalar_var = x
  [../]

  [./right]
    type = ScalarDirichletBC
    variable = diffused
    boundary = 3
    scalar_var = y
  [../]
[]

[Postprocessors]
  # to print the values of x, y into a file so we can plot it
  [./x]
    type = ScalarVariable
    variable = x
    execute_on = timestep
  [../]
  [./y]
    type = ScalarVariable
    variable = y
    execute_on = timestep
  [../]

  [./exact_x]
    type = PlotFunction
    function = exact_x_fn
    execute_on = timestep
  [../]
  # measure the error from exact solution in L2 norm
  [./l2err_x]
    type = ScalarL2Error
    variable = x
    function = exact_x_fn
  [../]
[]


[Executioner]
  type = Transient
  start_time = 0
  dt = 0.01
  num_steps = 10

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  file_base = out
  output_initial = true
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 80: Example 18: ImplicitODEx.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef IMPLICITODEX_H
#define IMPLICITODEX_H


#include "ODEKernel.h"

/**
 * The forward declaration is so that we can declare the validParams function
 * before we actually define the class... that way the definition isn't lost
 * at the bottom of the file.
 */

// Forward Declarations
class ImplicitODEx;

/**
 * validParams returns the parameters that this Kernel accepts / needs
 * The actual body of the function MUST be in the .C file.
 */
template<>
InputParameters validParams<ImplicitODEx>();

/**
 * ODE: x' = 3 * x + 2 * y
 */
class ImplicitODEx : public ODEKernel
{
public:
  /**
   * Constructor
   */
  ImplicitODEx(const std::string & name, InputParameters parameters);

protected:
  /**
   * Responsible for computing the residual
   */
  virtual Real computeQpResidual();

  /**
   * Responsible for computing the diagonal block of the preconditioning matrix.
   * This is essentially the partial derivative of the residual with respect to
   * the variable this kernel operates on ("u").
   *
   * Note that this can be an approximation or linearization.  In this case it's
   * not because the Jacobian of this operator is easy to calculate.
   */
  virtual Real computeQpJacobian();

  /**
   * Responsible for computing the off-diagonal block of the preconditioning matrix.
   * This is essentially the partial derivative of the residual with respect to
   * the variable that is coupled into this kernel.
   *
   * Note that this can be an approximation or linearization.  In this case it's
```

```
   * not because the Jacobian of this operator is easy to calculate.
   */
  virtual Real computeQpOffDiagJacobian(unsigned int jvar);

  /**
   * Needed for computing off-diagonal terms in Jacobian
   */
  unsigned int _y_var;

  /**
   * Coupled scalar variable values
   */
  VariableValue & _y;
};


#endif /* IMPLICITODEX_H */
```

Listing 81: Example 18: ImplicitODEx.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ImplicitODEx.h"

/**
 * This function defines the valid parameters for
 * this Kernel and their default values
 */
template<>
InputParameters validParams<ImplicitODEx>()
{
  InputParameters params = validParams<ODEKernel>();
  params.addCoupledVar("y", "variable Y coupled into this kernel");
  return params;
}

ImplicitODEx::ImplicitODEx(const std::string & name, InputParameters parameters) :
    // You must call the constructor of the base class first
    ODEKernel(name, parameters),
    // get the coupled variable number and values
    _y_var(coupledScalar("y")),
    _y(coupledScalarValue("y"))
{
}

Real
ImplicitODEx::computeQpResidual()
{
  // the term of the ODE without the time derivative term
  return -3. * _u[_i] - 2. * _y[_i];
}

Real
ImplicitODEx::computeQpJacobian()
{
  // dF/dx
  return -3.;
}

Real
ImplicitODEx::computeQpOffDiagJacobian(unsigned int jvar)
{
  if (jvar == _y_var)
    return -2.;         // dF/dy
  else
    return 0.;          // everything else
}
```

Listing 82: Example 18: ImplicitODEy.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef IMPLICITODEY_H
#define IMPLICITODEY_H

#include "ODEKernel.h"

/**
 * The forward declaration is so that we can declare the validParams function
 * before we actually define the class... that way the definition isn't lost
 * at the bottom of the file.
 */

// Forward Declarations
class ImplicitODEy;

/**
 * validParams returns the parameters that this Kernel accepts / needs
 * The actual body of the function MUST be in the .C file.
 */
template<>
InputParameters validParams<ImplicitODEy>();

/**
 * Kernel that implements the ODE for y-variable
 *
 * ODE: y' = 4 * x + y
 */
class ImplicitODEy : public ODEKernel
{
public:
  /**
   * Constructor
   */
  ImplicitODEy(const std::string & name, InputParameters parameters);

protected:
  /**
   * Responsible for computing the residual
   */
  virtual Real computeQpResidual();

  /**
   * Responsible for computing the diagonal block of the preconditioning matrix.
   * This is essentially the partial derivative of the residual with respect to
   * the variable this kernel operates on ("u").
   *
   * Note that this can be an approximation or linearization.  In this case it's
   * not because the Jacobian of this operator is easy to calculate.
   */
  virtual Real computeQpJacobian();

  /**
   * Responsible for computing the off-diagonal block of the preconditioning matrix.
   * This is essentially the partial derivative of the residual with respect to
   * the variable that is coupled into this kernel.
   */
```

```
  virtual Real computeQpOffDiagJacobian(unsigned int jvar);

  /**
   * Needed for computing off-diagonal terms in Jacobian
   */
  unsigned int _x_var;

  /**
   * Coupled scalar variable values
   */
  VariableValue & _x;
};


#endif /* IMPLICITODEY_H */
```

Listing 83: Example 18: ImplicitODEy.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ImplicitODEy.h"

/**
 * This function defines the valid parameters for
 * this Kernel and their default values
 */
template<>
InputParameters validParams<ImplicitODEy>()
{
  InputParameters params = validParams<ODEKernel>();
  params.addCoupledVar("x", "variable X coupled into this kernel");
  return params;
}

ImplicitODEy::ImplicitODEy(const std::string & name, InputParameters parameters) :
    // You must call the constructor of the base class first
    ODEKernel(name, parameters),
    // get the coupled variable number and values
    _x_var(coupledScalar("x")),
    _x(coupledScalarValue("x"))
{
}

Real
ImplicitODEy::computeQpResidual()
{
  // the term of the ODE without the time derivative term
  return -4 * _x[_i] - _u[_i];
}

Real
ImplicitODEy::computeQpJacobian()
{
  // dF/dy
  return -1.;
}

Real
ImplicitODEy::computeQpOffDiagJacobian(unsigned int jvar)
{
  if (jvar == _x_var)
    return -4.;         // dF/dx
  else
    return 0.;          // everything else
}
```

Listing 84: Example 18: ScalarDirichletBC.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef SCALARDIRICHLETBC_H
#define SCALARDIRICHLETBC_H

#include "NodalBC.h"

//Forward Declarations
class ScalarDirichletBC;

template<>
InputParameters validParams<ScalarDirichletBC>();

/**
 * Implements a Dirichlet BC where scalar variable is coupled in
 */
class ScalarDirichletBC : public NodalBC
{
public:

  /**
   * Factory constructor, takes parameters so that all derived classes can be built using the same
   * constructor.
   */
  ScalarDirichletBC(const std::string & name, InputParameters parameters);

protected:
  virtual Real computeQpResidual();

  /**
   * Holds the values of a coupled scalar variable.
   */
  VariableValue & _scalar_val;
};

#endif // SCALARDIRICHLETBC_H
```

Listing 85: Example 18: ScalarDirichletBC.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ScalarDirichletBC.h"

template<>
InputParameters validParams<ScalarDirichletBC>()
{
  InputParameters params = validParams<NodalBC>();
  // Here we are adding a parameter that will be extracted from the input file by the Parser
  params.addRequiredCoupledVar("scalar_var", "Value of the scalar variable");
  return params;
}

ScalarDirichletBC::ScalarDirichletBC(const std::string & name, InputParameters parameters) :
    NodalBC(name, parameters),

    /**
     * Get a reference to the coupled variable's values.
     */
    _scalar_val(coupledScalarValue("scalar_var"))
{}

Real
ScalarDirichletBC::computeQpResidual()
{
  // We coupled in a first order scalar variable, thus there is only one value in _scalar_val (and it is
       - big surprise - on index 0)
  return _u[_qp] - _scalar_val[0];
}
```

Listing 86: Example 18: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "AppFactory.h"

// Example 18 Includes
#include "ScalarDirichletBC.h"
#include "ImplicitODEx.h"
#include "ImplicitODEy.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}

void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerBoundaryCondition(ScalarDirichletBC);
  registerScalarKernel(ImplicitODEx);
  registerScalarKernel(ImplicitODEy);
}

void
ExampleApp::associateSyntax(Syntax & /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 19

Newton Damping

Listing 87: Example 19: ex19.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2

  xmin = 0.0
  xmax = 1.0
  nx = 10

  ymin = 0.0
  ymax = 1.0
  ny = 10
[]

[Variables]
  active = 'diffusion'

  [./diffusion]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  active = 'diff'

  [./diff]
    type = Diffusion
    variable = diffusion
  [../]
[]

[BCs]
  active = 'left right'

  [./left]
    type = DirichletBC
    variable = diffusion
    boundary = 1
    value = 3
  [../]

  [./right]
    type = DirichletBC
    variable = diffusion
    boundary = 2
    value = 1
  [../]
[]

[Dampers]
  # Use a constant damping parameter
  [./diffusion_damp]
    type = ConstantDamper
    variable = diffusion
    damping = 0.9
  [../]
[]

[Executioner]
  type = Steady

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


[]

[Outputs]
  file_base = out
```

```
  output_initial = true
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Example: 20

UserObjects

Listing 88: Example 20: ex20.i

```
[Mesh]
  file = two_squares.e
  dim = 2
[]

[Variables]
  [./u]
    initial_condition = 0.01
  [../]
[]

[Kernels]
  [./diff]
    type = ExampleDiffusion
    variable = u
  [../]
  [./td]
    type = TimeDerivative
    variable = u
  [../]
[]

[BCs]
  [./left]
    type = DirichletBC
    variable = u
    boundary = leftleft
    value = 0
  [../]
  [./right]
    type = DirichletBC
    variable = u
    boundary = rightright
    value = 1
  [../]
[]

[Materials]
  [./badm]
    type = BlockAverageDiffusionMaterial
    block = 'left right'
    block_average_userobject = bav
  [../]
[]

[UserObjects]
  [./bav]
    type = BlockAverageValue
    variable = u
    execute_on = timestep_begin
  [../]
[]

[Executioner]
  type = Transient
  num_steps = 10
  dt = 1

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  output_initial = true
  exodus = true
  [./console]
```

```
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 89: Example 20: BlockAverageValue.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef BLOCKAVERAGEVALUE_H
#define BLOCKAVERAGEVALUE_H

#include "ElementIntegralVariablePostprocessor.h"

// libmesh includes
#include "libmesh/mesh_tools.h"

//Forward Declarations
class BlockAverageValue;

template<>
InputParameters validParams<BlockAverageValue>();

/**
 * Computes the average value of a variable on each block
 */
class BlockAverageValue : public ElementIntegralVariablePostprocessor
{
public:
  BlockAverageValue(const std::string & name, InputParameters parameters);

  /**
   * Given a block ID return the average value for a variable on that block
   *
   * Note that accessor functions on UserObjects like this _must_ be const.
   * That is because the UserObject system returns const references to objects
   * trying to use UserObjects.  This is done for parallel correctness.
   *
   * @return The average value of a variable on that block.
   */
  Real averageValue(SubdomainID block) const;

  /**
   * This is called before execute so you can reset any internal data.
   */
  virtual void initialize();

  /**
   * Called on every "object" (like every element or node).
   * In this case, it is called at every quadrature point on every element.
   */
  virtual void execute();

  /**
   * Called when using threading.  You need to combine the data from "y"
   * into _this_ object.
   */
  virtual void threadJoin(const UserObject & y);

  /**
   * Called _once_ after execute has been called all all "objects".
   */
  virtual void finalize();
```

```
protected:
  // This map will hold the partial sums for each block
  std::map<SubdomainID, Real> _integral_values;

  // This map will hold the partial volume sums for each block
  std::map<SubdomainID, Real> _volume_values;

  // This map will hold our averages for each block
  std::map<SubdomainID, Real> _average_values;
};

#endif
```

Listing 90: Example 20: BlockAverageValue.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "BlockAverageValue.h"

// libmesh includes
#include "libmesh/mesh_tools.h"

template<>
InputParameters validParams<BlockAverageValue>()
{
  InputParameters params = validParams<ElementIntegralVariablePostprocessor>();

  // Since we are inheriting from a Postprocessor we override this to make sure
  // That MOOSE (and Peacock) know that this object is _actually_ a UserObject
  params.set<std::string>("built_by_action") = "add_user_object";

  return params;
}

BlockAverageValue::BlockAverageValue(const std::string & name, InputParameters parameters) :
    ElementIntegralVariablePostprocessor(name, parameters)
{
}

Real
BlockAverageValue::averageValue(SubdomainID block) const
{
  // Note that we can't use operator[] for a std::map in a const function!
  if (_average_values.find(block) != _average_values.end())
    return _average_values.find(block)->second;

  mooseError("Unknown block requested for average value!");

  return 0; // To satisfy compilers
}

void
BlockAverageValue::initialize()
{
  // Explicitly call the initialization routines for our base class
  ElementIntegralVariablePostprocessor::initialize();

  // Set averages to 0 for each block
  const std::set<SubdomainID> & blocks = _subproblem.mesh().meshSubdomains();

  for (std::set<SubdomainID>::const_iterator it = blocks.begin();
      it != blocks.end();
      ++it)
  {
    _integral_values[*it] = 0;
    _volume_values[*it] = 0;
    _average_values[*it] = 0;
  }
}

void
BlockAverageValue::execute()
```

```
{
  // Compute the integral on this element
  Real integral_value = computeIntegral();

  // Add that value to the others we've computed on this subdomain
  _integral_values[_current_elem->subdomain_id()] += integral_value;

  // Keep track of the volume of this block
  _volume_values[_current_elem->subdomain_id()] += _current_elem_volume;
}

void
BlockAverageValue::threadJoin(const UserObject & y)
{
  ElementIntegralVariablePostprocessor::threadJoin(y);

  // We are joining with another class like this one so do a cast so we can get to it's data
  const BlockAverageValue & bav = dynamic_cast<const BlockAverageValue &>(y);

  for (std::map<SubdomainID, Real>::const_iterator it = bav._integral_values.begin();
      it != bav._integral_values.end();
      ++it)
    _integral_values[it->first] += it->second;

  for (std::map<SubdomainID, Real>::const_iterator it = bav._volume_values.begin();
      it != bav._volume_values.end();
      ++it)
    _volume_values[it->first] += it->second;

  for (std::map<SubdomainID, Real>::const_iterator it = bav._average_values.begin();
      it != bav._average_values.end();
      ++it)
    _average_values[it->first] += it->second;
}

void
BlockAverageValue::finalize()
{
  // Loop over the integral values and sum them up over the processors
  for (std::map<SubdomainID, Real>::iterator it = _integral_values.begin();
      it != _integral_values.end();
      ++it)
    gatherSum(it->second);

  // Loop over the volumes and sum them up over the processors
  for (std::map<SubdomainID, Real>::iterator it = _volume_values.begin();
      it != _volume_values.end();
      ++it)
    gatherSum(it->second);

  // Now everyone has the correct data so everyone can compute the averages properly:
  for (std::map<SubdomainID, Real>::iterator it = _average_values.begin();
      it != _average_values.end();
      ++it)
  {
    SubdomainID id = it->first;
    _average_values[id] = _integral_values[id] / _volume_values[id];
  }
}
```

Listing 91: Example 20: BlockAverageDiffusionMaterial.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef BLOCKAVERAGEDIFFUSIONMATERIAL_H
#define BLOCKAVERAGEDIFFUSIONMATERIAL_H

#include "Material.h"
#include "BlockAverageValue.h"

//Forward Declarations
class BlockAverageDiffusionMaterial;

template<>
InputParameters validParams<BlockAverageDiffusionMaterial>();

class BlockAverageDiffusionMaterial : public Material
{
public:
  BlockAverageDiffusionMaterial(const std::string & name,
                  InputParameters parameters);

protected:
  virtual void computeQpProperties();

private:
  /**
   * This is the member reference that will hold the computed values
   * for the Real value property in this class.
   */
  MaterialProperty<Real> & _diffusivity;

  /**
   * A member reference that will hold onto a UserObject
   * of type BlockAverageValue for us to be able to query
   * the average value of a variable on each block.
   *
   * NOTE: UserObject references are _const_!
   */
  const BlockAverageValue & _block_average_value;
};

#endif //BLOCKAVERAGEDIFFUSIONMATERIAL_H
```

Listing 92: Example 20: BlockAverageDiffusionMaterial.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "BlockAverageDiffusionMaterial.h"

template<>
InputParameters validParams<BlockAverageDiffusionMaterial>()
{
  InputParameters params = validParams<Material>();

  // UserObjectName is the MOOSE type used for getting the name of a UserObject from the input file
  params.addRequiredParam<UserObjectName>("block_average_userobject", "The name of the UserObject that
      is going to be computing the average value of a variable on each block");

  return params;
}

BlockAverageDiffusionMaterial::BlockAverageDiffusionMaterial(const std::string & name,
                                InputParameters parameters) :
    Material(name, parameters),

    // Declare that this material is going to provide a Real
    // valued property named "diffusivity" that Kernels can use.
    _diffusivity(declareProperty<Real>("diffusivity")),

    // When getting a UserObject from the input file pass the name
    // of the UserObjectName _parameter_
    // Note that getUserObject returns a _const reference_ of the type in < >
    _block_average_value(getUserObject<BlockAverageValue>("block_average_userobject"))
{}

void
BlockAverageDiffusionMaterial::computeQpProperties()
{
  // We will compute the diffusivity based on the average value of the variable on each block.

  // We'll get that value from a UserObject that is computing it for us.

  // To get the current block number we're going to query the "subdomain_id()" of the current element
  _diffusivity[_qp] = 0.5 * _block_average_value.averageValue(_current_elem->subdomain_id());
}
```

Listing 93: Example 20: ExampleDiffusion.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEDIFFUSION_H
#define EXAMPLEDIFFUSION_H

#include "Diffusion.h"

//Forward Declarations
class ExampleDiffusion;

/**
 * validParams returns the parameters that this Kernel accepts / needs
 * The actual body of the function MUST be in the .C file.
 */
template<>
InputParameters validParams<ExampleDiffusion>();

class ExampleDiffusion : public Diffusion
{
public:

  ExampleDiffusion(const std::string & name,
                   InputParameters parameters);

protected:
  virtual Real computeQpResidual();
  virtual Real computeQpJacobian();

  /**
   * This MooseArray will hold the reference we need to our
   * material property from the Material class
   */
  MaterialProperty<Real> & _diffusivity;
};
#endif //EXAMPLEDIFFUSION_H
```

Listing 94: Example 20: ExampleDiffusion.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleDiffusion.h"

/**
 * This function defines the valid parameters for
 * this Kernel and their default values
 */
template<>
InputParameters validParams<ExampleDiffusion>()
{
  InputParameters params = validParams<Diffusion>();
  return params;
}


ExampleDiffusion::ExampleDiffusion(const std::string & name,
                                   InputParameters parameters) :
    Diffusion(name,parameters),
    _diffusivity(getMaterialProperty<Real>("diffusivity"))
{}

Real
ExampleDiffusion::computeQpResidual()
{
  // We're dereferencing the _diffusivity pointer to get to the
  // material properties vector... which gives us one property
  // value per quadrature point.

  // Also... we're reusing the Diffusion Kernel's residual
  // so that we don't have to recode that.
  return _diffusivity[_qp]*Diffusion::computeQpResidual();
}

Real
ExampleDiffusion::computeQpJacobian()
{
  // We're dereferencing the _diffusivity pointer to get to the
  // material properties vector... which gives us one property
  // value per quadrature point.

  // Also... we're reusing the Diffusion Kernel's residual
  // so that we don't have to recode that.
  return _diffusivity[_qp]*Diffusion::computeQpJacobian();
}
```

Listing 95: Example 20: ExampleApp.C

```
#include "ExampleApp.h"
#include "Moose.h"
#include "AppFactory.h"

// Example Includes
#include "BlockAverageDiffusionMaterial.h"
#include "BlockAverageValue.h"
#include "ExampleDiffusion.h"

template<>
InputParameters validParams<ExampleApp>()
{
  InputParameters params = validParams<MooseApp>();
  return params;
}

ExampleApp::ExampleApp(const std::string & name, InputParameters parameters) :
    MooseApp(name, parameters)
{
  srand(processor_id());

  Moose::registerObjects(_factory);
  ExampleApp::registerObjects(_factory);

  Moose::associateSyntax(_syntax, _action_factory);
  ExampleApp::associateSyntax(_syntax, _action_factory);
}

ExampleApp::~ExampleApp()
{
}



void
ExampleApp::registerApps()
{
  registerApp(ExampleApp);
}

void
ExampleApp::registerObjects(Factory & factory)
{
  registerMaterial(BlockAverageDiffusionMaterial);
  registerKernel(ExampleDiffusion);

  // This is how to register a UserObject
  registerUserObject(BlockAverageValue);
}

void
ExampleApp::associateSyntax(Syntax & /*syntax*/, ActionFactory & /*action_factory*/)
{
}
```

Example: 21

Debugging

Listing 96: Example 21: ex21.i

```
[Mesh]
  file = reactor.e
  # Let's assign human friendly names to the blocks on the fly
  block_id = '1 2'
  block_name = 'fuel deflector'

  boundary_id = '4 5'
  boundary_name = 'bottom top'
[]

[Variables]
  [./diffused]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.5
  [../]

  [./convected]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.0
  [../]
[]

[Kernels]
  # This Kernel consumes a real-gradient material property from the active material
  [./convection]
    type = Convection
    variable = convected
  [../]

  [./diff_convected]
    type = Diffusion
    variable = convected
  [../]

  [./example_diff]
    # This Kernel uses "diffusivity" from the active material
    type = ExampleDiffusion
    variable = diffused
  [../]

  [./time_deriv_diffused]
    type = TimeDerivative
    variable = diffused
  [../]

  [./time_deriv_convected]
    type = TimeDerivative
    variable = convected
  [../]
[]

[BCs]
  [./bottom_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'bottom'
    value = 0
  [../]

  [./top_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'top'
    value = 5
  [../]

  [./bottom_convected]
```

```
    type = DirichletBC
    variable = convected
    boundary = 'bottom'
    value = 0
  [../]

  [./top_convected]
    type = NeumannBC
    variable = convected
    boundary = 'top'
    value = 1
  [../]
[]

[Materials]
  [./example]
    type = ExampleMaterial
    block = 'fuel'
    diffusion_gradient = 'diffused'

    # Approximate Parabolic Diffusivity
    independent_vals = '0 0.25 0.5 0.75 1.0'
    dependent_vals = '1e-2 5e-3 1e-3 5e-3 1e-2'
  [../]

  [./example1]
    type = ExampleMaterial
    block = 'deflector'
    diffusion_gradient = 'diffused'

    # Constant Diffusivity
    independent_vals = '0 1.0'
    dependent_vals = '1e-1 1e-1'
  [../]
[]

[Executioner]
  type = Transient

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'


  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'

  dt = 0.1
  num_steps = 10
[]

[Outputs]
  file_base = out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    linear_residuals = true
  [../]
[]
```

Listing 97: Example 21: ExampleDiffusion.h

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#ifndef EXAMPLEDIFFUSION_H
#define EXAMPLEDIFFUSION_H

#include "Diffusion.h"

//Forward Declarations
class ExampleDiffusion;

/**
 * validParams returns the parameters that this Kernel accepts / needs
 * The actual body of the function MUST be in the .C file.
 */
template<>
InputParameters validParams<ExampleDiffusion>();

class ExampleDiffusion : public Diffusion
{
public:

  ExampleDiffusion(const std::string & name,
                   InputParameters parameters);

protected:
  virtual Real computeQpResidual();
  virtual Real computeQpJacobian();

  /**
   * THIS IS AN ERROR ON PURPOSE!
   *
   * The "&" is missing here!
   *
   * Do NOT copy this line of code!
   */
  MaterialProperty<Real> _diffusivity;
};
#endif //EXAMPLEDIFFUSION_H
```

Listing 98: Example 21: ExampleDiffusion.C

```
/****************************************************************/
/*               DO NOT MODIFY THIS HEADER                      */
/* MOOSE - Multiphysics Object Oriented Simulation Environment  */
/*                                                              */
/*           (c) 2010 Battelle Energy Alliance, LLC             */
/*                   ALL RIGHTS RESERVED                        */
/*                                                              */
/*          Prepared by Battelle Energy Alliance, LLC           */
/*            Under Contract No. DE-AC07-05ID14517              */
/*            With the U. S. Department of Energy               */
/*                                                              */
/*            See COPYRIGHT for full restrictions               */
/****************************************************************/

#include "ExampleDiffusion.h"

/**
 * This function defines the valid parameters for
 * this Kernel and their default values
 */
template<>
InputParameters validParams<ExampleDiffusion>()
{
  InputParameters params = validParams<Diffusion>();
  return params;
}


ExampleDiffusion::ExampleDiffusion(const std::string & name,
                                   InputParameters parameters) :
    Diffusion(name,parameters),
    _diffusivity(getMaterialProperty<Real>("diffusivity"))
{}

Real
ExampleDiffusion::computeQpResidual()
{
  // We're dereferencing the _diffusivity pointer to get to the
  // material properties vector... which gives us one property
  // value per quadrature point.

  // Also... we're reusing the Diffusion Kernel's residual
  // so that we don't have to recode that.
  return _diffusivity[_qp]*Diffusion::computeQpResidual();
}

Real
ExampleDiffusion::computeQpJacobian()
{
  // We're dereferencing the _diffusivity pointer to get to the
  // material properties vector... which gives us one property
  // value per quadrature point.

  // Also... we're reusing the Diffusion Kernel's residual
  // so that we don't have to recode that.
  return _diffusivity[_qp]*Diffusion::computeQpJacobian();
}
```