DIDASKO : Tutorial for Trilinos

Version 1.0

Generated by Doxygen 1.5.9

Thu Dec 17 11:06:28 2009

Contents

Chapter 1

DIDASKO : Tutorial for Trilinos

1.1 Table of Contents

1.2 Introduction

DIDASKO, the tutorial of Trilinos, offers a quick overview of Trilinos. The reader can find a variety of examples on several Trilinos packages. Each example is documented with comments and suggestions, some in the source file itself, and some others in the PDF document. The PDF version of the tutorial can be found in the Trilinos web page.

Usually, a Trilinos package interoperates with other Trilinos packages (and with external packages as well). To be used efficiently, the examples of DIDASKO would require the following compilation flags:

```
--enable-epetra --enable-aztecoo --enable-amesos --enable-anasazi \
--enable-ifpack --enable-ml --enable-nox --enable-teuchos \
--enable-triutils
```

Most of the required packages are enabled by default. No example can effectively run without Epetra, and most require AztecOO and Triutils (this latter to generate the linear system matrix).

All the Didasko examples include, as first header file, the file

```
#include "Didasko_ConfigDefs.h"
```

Then, the code checks for installed libraries, like for instance

```
#if defined(HAVE_DIDASKO_EPETRA)
```

In the following, we report the source code of each of the examples. Packages are reported in alphabetical order.

1.3 Amesos

- Basic usage of Amesos: amesos_ex1

1.4 Anasazi Examples

- Computation of the lowest eigenvalues and the corresponding eigenvectors of a symmetric matrix using a Block Krylov Schur solver: anasazi_ex1
- Computation of the lowest eigenvalues and the corresponding eigenvectors of a symmetric matrix using a Block Davidson solver: anasazi_ex2
- Computation of the lowest eigenvalues and the corresponding eigenvectors of a nonsymmetric matrix using a Block Krylov Schur solver: anasazi_ex3
- Computation of the lowest eigenvalues and the corresponding eigenvectors of a symmetric matrix using an LOBPCG solver: anasazi_ex4

1.5 Epetra Examples

- Basic usage of communicators: epetra_ex1
- Definition of Epetra_Map objects: epetra_ex2
- Definition of vectors: epetra_ex3
- Definition of serial dense vectors: epetra_ex4
- Basic operations on vectors. Work on the element vectors: epetra_ex5

- Use of ExtractView of Epetra_Vector: epetra_ex6
- Example of Epetra_MultiVector; use of ExtractView on MultiVector: epetra_ex7
- Epetra_Vectors in View mode; use of ResetView: epetra_ex8
- Epetra_Export classes: epetra_ex9
- Definition of dense serial matrices: epetra_ex10
- Basic definition of communicators: epetra_ex11
- Creation of a tridiagonal matrix: epetra_ex12
- Simple FE code: epetra_ex13
- Print out some information about an Epetra_CrsMatrix: epetra_ex14
- Output an Epetra_CrsMatrix in MATLAB format: epetra_ex15
- Output a vector in MATLAB format: epetra_ex16
- Construct an Epetra_VrbMatrix: epetra_ex17
- Basic operations on vectors: epetra_ex18
- Create an Epetra_CrsMatrix corresponding to a 2D Laplacian problem on a cartesian mesh: epetra_ex19
- Example of use of Epetra_Time and Epetra_Flops: epetra_ex20
- Use of Epetra_Operator: epetra_ex21, epetra_ex22
- Use of Epetra_FECrsMatrix: epetra_ex23
- Show the usage of RowMap, ColumnMap, RangeMap and DomainMap: epetra_ex24

1.6 EpetraExt Examples

- Generate a matrix using triutils and redistribute with Zoltan: epetraext_ex1
- Generate a linearproblem using triutils and redistribute with Zoltan: epetraext_ex2

1.7 AztecOO Examples

- Simple Jacobi preconditioner: aztecoo_ex1
- AztecOO as a preconditioner for AztecOO: aztecoo_ex2
- AztecOO CG and ICC(0) factorization as preconditioner: aztecoo_ex3

1.8 IFPACK Examples

- Using IFPACK factorizations as Aztec's preconditioners: ifpack_ex1, ifpack_ex2

1.9 ML Examples

- Two-level domain decomposition preconditioner with AztecOO and ML: ml_ex1
- Use of ML as a black-box smoothed aggregation preconditioner: ml_ex2

1.10 NOX Examples

- Simple nonlinear problem: nox_ex1
- Simple nonlinear PDE problem: nox_ex2

1.11 Teuchos Examples

- Teuchos dense matrices: teuchos_ex1
- Teuchos interface to BLAS: teuchos_ex2
- Teuchos interface to LAPACK: teuchos_ex3
- Teuchos parameters list: teuchos_ex4
- Teuchos RCP: teuchos_ex5
- Teuchos time monitor: teuchos_ex6
- Teuchos command line processor: teuchos_ex7

1.12 Triutils Examples

- Read a matrix in HB format: triutils_ex1
- Basic usage of CommandLineParser: triutils_ex2

Chapter 2

amesos_ex1

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_AMESOS) && defined(HAVE_
      DIDASKO_TEUCHOS) && defined(HAVE_DIDASKO_TRIUTILS)

#include "Epetra_ConfigDefs.h"
#include "Amesos_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Amesos.h"
#include "Trilinos_Util_CrsMatrixGallery.h"

using namespace Trilinos_Util;

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // initialize an Gallery object
  CrsMatrixGallery Gallery("laplace_2d", Comm);
  Gallery.Set("problem_size", 100); //must be a square number

  // get pointers to the linear problem, containing matrix, LHS and RHS.
  // if you need to access them, you can for example uncomment the following
  // code:
```

```
  // Epetra_CrsMatrix* Matrix = Gallery.GetMatrix();
  // Epetra_MultiVector* LHS = Gallery.GetStartingSolution();
  // Epetra_MultiVector* RHS = Gallery.GetRHS();
  //
  // NOTE: StartingSolution and RHS are pointers to Gallery's internally stored
  // vectors. Using StartingSolution and RHS, we can verify the residual
  // after the solution of the linear system. However, users may define as well
  // their own vectors for solution and RHS.

  Epetra_LinearProblem* Problem = Gallery.GetLinearProblem();

  // initialize Amesos solver:
  // 'Solver' is the pointer to the Amesos solver
  // (note the use of the base class Amesos_BaseSolver)
  Amesos_BaseSolver* Solver;
  // Amesos_Factory is the function class used to create the solver.
  // This class contains no data.
  Amesos Amesos_Factory;

  // empty parameter list
  Teuchos::ParameterList List;

  // may also try: "Amesos_Umfpack", "Amesos_Lapack", ...
  string SolverType = "Amesos_Klu";

  Solver = Amesos_Factory.Create(SolverType, *Problem);
  // Amesos_Factory returns 0 is the selected solver is not
  // available
  assert (Solver);

  // start solving
  Solver->SymbolicFactorization();
  Solver->NumericFactorization();
  Solver->Solve();

  // verify that residual is really small
  double residual, diff;

  Gallery.ComputeResidual(&residual);
  Gallery.ComputeDiffBetweenStartingAndExactSolutions(&diff);

  if( Comm.MyPID() == 0 ) {
    cout << "||b-Ax||_2 = " << residual << endl;
    cout << "||x_exact - x||_2 = " << diff << endl;
  }

  // delete Solver
  delete Solver;

  if (residual > 1e-5)
    exit(EXIT_FAILURE);

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  exit(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-teuchos\n"
       "--enable-triutils\n"
       "--enable-amesos");

  return 0;
}

#endif
```

Chapter 3

anasazi_ex1

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ // gmail.com)
//
// ***********************************************************************
// @HEADER

// Compute the smallest magnitude eigenvalues and corresponding eigenvectors usin
      g
// Anasazi::BlockKrylovSchur

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_ANASAZI) && defined(HAVE
      _DIDASKO_TEUCHOS) && defined(HAVE_DIDASKO_TRIUTILS)

#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif

#include "Epetra_MultiVector.h"
#include "Trilinos_Util_CrsMatrixGallery.h"
#include "AnasaziBasicEigenproblem.hpp"
#include "AnasaziEpetraAdapter.hpp"
#include "AnasaziBlockKrylovSchurSolMgr.hpp"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  // Initialize MPI and setup an Epetra communicator
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  // If we aren't using MPI, then setup a serial communicator.
  Epetra_SerialComm Comm;
#endif

  // Some typedefs for oft-used data types
```

```
typedef Epetra_MultiVector MV;
typedef Epetra_Operator OP;
typedef Anasazi::MultiVecTraits<double, Epetra_MultiVector> MVT;

bool ierr;

// Get our processor ID (0 if running serial)
int MyPID = Comm.MyPID();

// Verbose flag: only processor 0 should print
bool verbose = (MyPID==0);

// Initialize a Gallery object, from which we will select a matrix
// Select a 2-D laplacian, of order 100
Trilinos_Util::CrsMatrixGallery Gallery("laplace_2d", Comm);
Gallery.Set("problem_size", 100);

// Say hello and print some information about the gallery matrix
if (verbose) {
  cout << "Anasazi Example: Block Kyrlov Schur" << endl;
  cout << "Problem info:" << endl;
  cout << Gallery;
  cout << endl;
}

// Setup some more problem/solver parameters:

// Number of desired eigenvalues
int nev = 4;

// Block size
int blocksize = 4;


// Get a pointer to the system matrix, inside of a Teuchos::RCP
// The Teuchos::RCP is a reference counting pointer that handles
// garbage collection for us, so that we can perform memory allocation without
// having to worry about freeing memory manually.
Teuchos::RCP<Epetra_CrsMatrix> A = Teuchos::rcp( Gallery.GetMatrix(), false );

// Create an Anasazi MultiVector, based on Epetra MultiVector
const Epetra_Map * Map = &(A->RowMap());
Teuchos::RCP<Epetra_MultiVector> ivec =
  Teuchos::rcp( new Epetra_MultiVector(*Map,blocksize) );

// Fill it with random numbers
ivec->Random();


// Setup the eigenproblem, with the matrix A and the initial vectors ivec
Teuchos::RCP< Anasazi::BasicEigenproblem<double,MV,OP> > MyProblem =
  Teuchos::rcp( new Anasazi::BasicEigenproblem<double,MV,OP>(A, ivec) );

// The 2-D laplacian is symmetric. Specify this in the eigenproblem.
MyProblem->setHermitian(true);

// Specify the desired number of eigenvalues
MyProblem->setNEV( nev );

// Signal that we are done setting up the eigenvalue problem
ierr = MyProblem->setProblem();
```

```
  // Check the return from setProblem(). If this is true, there was an
  // error. This probably means we did not specify enough information for
  // the eigenproblem.
  assert(ierr == true);

  // Specify the verbosity level. Options include:
  // Anasazi::Errors
  //   This option is always set
  // Anasazi::Warnings
  //   Warnings (less severe than errors)
  // Anasazi::IterationDetails
  //   Details at each iteration, such as the current eigenvalues
  // Anasazi::OrthoDetails
  //   Details about orthogonality
  // Anasazi::TimingDetails
  //   A summary of the timing info for the solve() routine
  // Anasazi::FinalSummary
  //   A final summary
  // Anasazi::Debug
  //   Debugging information
  int verbosity = Anasazi::Warnings + Anasazi::Errors + Anasazi::FinalSummary + A
      nasazi::TimingDetails;

  // Choose which eigenvalues to compute
  // Choices are:
  // LM - target the largest magnitude  [default]
  // SM - target the smallest magnitude
  // LR - target the largest real
  // SR - target the smallest real
  // LI - target the largest imaginary
  // SI - target the smallest imaginary
  std::string which("SM");

  // Create the parameter list for the eigensolver
  Teuchos::ParameterList MyPL;
  MyPL.set( "Verbosity", verbosity );
  MyPL.set( "Which", which );
  MyPL.set( "Block Size", blocksize );
  MyPL.set( "Num Blocks", 20 );
  MyPL.set( "Maximum Restarts", 100 );
  MyPL.set( "Convergence Tolerance", 1.0e-8 );

  // Create the Block Krylov Schur solver
  // This takes as inputs the eigenvalue problem and the solver parameters
  Anasazi::BlockKrylovSchurSolMgr<double,MV,OP>
    MyBlockKrylovSchur(MyProblem, MyPL );

  // Solve the eigenvalue problem, and save the return code
  Anasazi::ReturnType solverreturn = MyBlockKrylovSchur.solve();

  // Check return code of the solver: Unconverged, Failed, or OK
  switch (solverreturn) {

  // UNCONVERGED
  case Anasazi::Unconverged:
    if (verbose)
      cout << "Anasazi::BlockKrylovSchur::solve() did not converge!" << endl;
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    return 0;
    break;
```

```
// CONVERGED
case Anasazi::Converged:
  if (verbose)
    cout << "Anasazi::BlockKrylovSchur::solve() converged!" << endl;
  break;
}

// Get eigensolution struct
Anasazi::Eigensolution<double, Epetra_MultiVector> sol = MyProblem->getSolution
    ();

// Get the number of eigenpairs returned
int numev = sol.numVecs;

// Get eigenvectors
Teuchos::RCP<Epetra_MultiVector> evecs = sol.Evecs;

// Get eigenvalues
std::vector<Anasazi::Value<double> > evals = sol.Evals;

// Test residuals
// Generate a (numev x numev) dense matrix for the eigenvalues...
// This matrix is automatically initialized to zero
Teuchos::SerialDenseMatrix<int, double> D(numev, numev);

// Add the eigenvalues on the diagonals (only the real part since problem is He
    rmitian)
for (int i=0; i<numev; i++) {
  D(i,i) = evals[i].realpart;
}

// Generate a multivector for the product of the matrix and the eigenvectors
Epetra_MultiVector R(*Map, numev);

// R = A*evecs
A->Apply( *evecs, R );

// R -= evecs*D
//    = A*evecs - evecs*D
MVT::MvTimesMatAddMv( -1.0, *evecs, D, 1.0, R );

// Compute the 2-norm of each vector in the MultiVector
// and store them to a std::vector<double>
std::vector<double> normR(numev);
MVT::MvNorm( R, normR );

// Output results to screen
if(verbose) {
  cout << scientific << showpos << setprecision(6) << showpoint;
  cout << "*******************************************************\n"
       << "           Results (outside of eigensolver)           \n"
       << "-------------------------------------------------------\n"
       << "  evals\t\t\tnormR\n"
       << "-------------------------------------------------------\n";
  for( int i=0 ; i<numev ; ++i ) {
    cout << "  " << evals[i].realpart << "\t\t" << normR[i] << endl;
  }
  cout << "*******************************************************\n" << endl;
}
```

```
#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
#endif

  puts("Please configure Didasko with:");
  puts("--enable-epetra");
  puts("--enable-teuchos");
  puts("--enable-triutils");
  puts("--enable-anasazi");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 4

anasazi_ex2

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ // gmail.com)
//
// ************************************************************************
// @HEADER

// Compute the smallest magnitude eigenvalues and the corresponding eigenvectors
      using
// Anasazi::BlockDavidson

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_ANASAZI) && defined(HAVE
      _DIDASKO_TEUCHOS) && defined(HAVE_DIDASKO_TRIUTILS)

#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif

#include "Epetra_MultiVector.h"
#include "Trilinos_Util_CrsMatrixGallery.h"
#include "AnasaziBasicEigenproblem.hpp"
#include "AnasaziEpetraAdapter.hpp"
#include "AnasaziBlockDavidsonSolMgr.hpp"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  // Initialize MPI and setup an Epetra communicator
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  // If we aren't using MPI, then setup a serial communicator.
  Epetra_SerialComm Comm;
#endif

  // Some typedefs for oft-used data types
```

```
typedef Epetra_MultiVector MV;
typedef Epetra_Operator OP;
typedef Anasazi::MultiVecTraits<double, Epetra_MultiVector> MVT;

bool ierr;

// Get our processor ID (0 if running serial)
int MyPID = Comm.MyPID();

// Verbose flag: only processor 0 should print
bool verbose = (MyPID==0);

// Initialize a Gallery object, from which we will select a matrix
// Select a 2-D laplacian, of order 100
Trilinos_Util::CrsMatrixGallery Gallery("laplace_2d", Comm);
Gallery.Set("problem_size", 100);

// Say hello and print some information about the gallery matrix
if (verbose) {
  cout << "Anasazi Example: Block Davidson" << endl;
  cout << "Problem info:" << endl;
  cout << Gallery;
  cout << endl;
}


// Setup some more problem/solver parameters:

// Number of desired eigenvalues
int nev = 4;

// Block size
int blocksize = 8;


// Get a pointer to the system matrix, inside of a Teuchos::RCP
// The Teuchos::RCP is a reference counting pointer that handles
// garbage collection for us, so that we can perform memory allocation without
// having to worry about freeing memory manually.
Teuchos::RCP<Epetra_CrsMatrix> A = Teuchos::rcp( Gallery.GetMatrix(), false );


// Create an Anasazi MultiVector, based on Epetra MultiVector
const Epetra_Map * Map = &(A->RowMap());
Teuchos::RCP<Epetra_MultiVector> ivec =
  Teuchos::rcp( new Epetra_MultiVector(*Map,blocksize) );

// Fill it with random numbers
ivec->Random();


// Setup the eigenproblem, with the matrix A and the initial vectors ivec
Teuchos::RCP< Anasazi::BasicEigenproblem<double,MV,OP> > MyProblem =
  Teuchos::rcp( new Anasazi::BasicEigenproblem<double,MV,OP>(A, ivec) );

// The 2-D laplacian is symmetric. Specify this in the eigenproblem.
MyProblem->setHermitian(true);

// Specify the desired number of eigenvalues
MyProblem->setNEV( nev );

// Signal that we are done setting up the eigenvalue problem
```

```
  ierr = MyProblem->setProblem();

  // Check the return from setProblem(). If this is true, there was an
  // error. This probably means we did not specify enough information for
  // the eigenproblem.
  assert(ierr);

  // Specify the verbosity level. Options include:
  // Anasazi::Errors
  //   This option is always set
  // Anasazi::Warnings
  //   Warnings (less severe than errors)
  // Anasazi::IterationDetails
  //   Details at each iteration, such as the current eigenvalues
  // Anasazi::OrthoDetails
  //   Details about orthogonality
  // Anasazi::TimingDetails
  //   A summary of the timing info for the solve() routine
  // Anasazi::FinalSummary
  //   A final summary
  // Anasazi::Debug
  //   Debugging information
  int verbosity = Anasazi::Warnings + Anasazi::Errors + Anasazi::FinalSummary;

  // Choose which eigenvalues to compute.
  // Choices are:
  // LM - target the largest magnitude  [default]
  // SM - target the smallest magnitude
  // LR - target the largest real
  // SR - target the smallest real
  std::string which("SM");

  // Create the parameter list for the eigensolver
  Teuchos::ParameterList MyPL;
  MyPL.set( "Verobsity", verbosity );
  MyPL.set( "Which", which );
  MyPL.set( "Block Size", blocksize );
  MyPL.set( "Num Blocks", 4       );
  MyPL.set( "Maximum Iters", 1000    );
  MyPL.set( "Convergence Tolerance", 1.0e-8 );

  // Create the Block Davidson solver
  // This takes as inputs the eigenvalue problem that we constructed and the solv
      er parameters
  Anasazi::BlockDavidsonSolMgr<double,MV,OP> MyBlockDavidson(MyProblem, MyPL );

  // Solve the eigenvalue problem, and save the return code
  Anasazi::ReturnType solverreturn = MyBlockDavidson.solve();

  // Check return code of the solver: Unconverged, Failed, or OK
  switch (solverreturn) {

  // UNCONVERGED
  case Anasazi::Unconverged:
    if (verbose)
      cout << "Anasazi::BlockDavidson::solve() did not converge!" << endl;
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    return 0;
    break;
```

```
  // OK
  case Anasazi::Converged:
    if (verbose)
      cout << "Anasazi::BlockDavidson::solve() converged!" << endl;
    break;
  }

// Get eigensolution struct
  Anasazi::Eigensolution<double, Epetra_MultiVector> sol = MyProblem->getSolution
      ();

  // Get the number of eigenpairs returned
  int numev = sol.numVecs;

  // Get eigenvectors
  Teuchos::RCP<Epetra_MultiVector> evecs = sol.Evecs;

  // Get eigenvalues
  std::vector<Anasazi::Value<double> > evals = sol.Evals;

  // Test residuals
  // Generate a (numev x numev) dense matrix for the eigenvalues...
  // This matrix is automatically initialized to zero
  Teuchos::SerialDenseMatrix<int, double> D(numev, numev);

  // Add the eigenvalues on the diagonals
  for (int i=0; i<numev; i++) {
    D(i,i) = evals[i].realpart;
  }

  // Generate a multivector for the product of the matrix and the eigenvectors
  Epetra_MultiVector R(*Map, numev);

  // R = A*evecs
  A->Apply( *evecs, R );

  // R -= evecs*D
  //    = A*evecs - evecs*D
  MVT::MvTimesMatAddMv( -1.0, *evecs, D, 1.0, R );

  // Compute the 2-norm of each vector in the MultiVector
  // and store them to a std::vector<double>
  std::vector<double> normR(numev);
  MVT::MvNorm( R, normR );

  // Output results to screen
  if(verbose) {
    cout << scientific << showpos << setprecision(6) << showpoint;
    cout << "*******************************************************\n"
         << "           Results (outside of eigensolver)           \n"
         << "-------------------------------------------------------\n"
         << "  evals\t\t\tnormR\n"
         << "-------------------------------------------------------\n";
    for( int i=0 ; i<numev ; ++i ) {
      cout << "  " << evals[i].realpart << "\t\t" << normR[i] << endl;
    }
    cout << "*******************************************************\n" << endl;
  }

  // Perform MPI cleanup, if we are using MPI
#ifdef HAVE_MPI
  MPI_Finalize();
```

```
#endif

  return(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
#endif

  puts("Please configure Didasko with:");
  puts("--enable-epetra");
  puts("--enable-teuchos");
  puts("--enable-triutils");
  puts("--enable-anasazi");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 5

anasazi_ex3

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ // gmail.com)
//
// ***********************************************************************
// @HEADER

// Compute the lowest eigenvalues and corresponding eigenvectors using
// Anasazi::BlockKrylovSchur

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_ANASAZI) && defined(HAVE
      _DIDASKO_TEUCHOS) && defined(HAVE_DIDASKO_TRIUTILS)

#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif

#include "Epetra_MultiVector.h"
#include "Trilinos_Util_CrsMatrixGallery.h"
#include "AnasaziBasicEigenproblem.hpp"
#include "AnasaziEpetraAdapter.hpp"
#include "AnasaziBlockKrylovSchurSolMgr.hpp"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  // Initialize MPI and setup an Epetra communicator
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  // If we aren't using MPI, then setup a serial communicator.
  Epetra_SerialComm Comm;
#endif

  // Some typedefs for oft-used data types
  typedef Epetra_MultiVector MV;
```

```
typedef Epetra_Operator OP;
typedef Anasazi::MultiVecTraits<double, MV> MVT;
typedef Anasazi::OperatorTraits<double, MV, OP> OPT;

bool ierr;

// Get our processor ID (0 if running serial)
int MyPID = Comm.MyPID();

// Verbose flag: only processor 0 should print
bool verbose = (MyPID==0);

// Initialize a Gallery object, from which we will select a matrix
// Select recirc_2d, of order 100
Trilinos_Util::CrsMatrixGallery Gallery("recirc_2d", Comm);
Gallery.Set("problem_size", 100);

// Say hello and print some information about the gallery matrix
if (verbose) {
  cout << "Anasazi Example: Block Kyrlov Schur" << endl;
  cout << "Problem info:" << endl;
  cout << Gallery;
  cout << endl;
}

// Setup some more problem/solver parameters:

// Number of desired eigenvalues
int nev = 4;

// Block size
int blocksize = 4;


// Get a pointer to the system matrix, inside of a Teuchos::RCP
// The Teuchos::RCP is a reference counting pointer that handles
// garbage collection for us, so that we can perform memory allocation without
// having to worry about freeing memory manually.
Teuchos::RCP<Epetra_CrsMatrix> A = Teuchos::rcp( Gallery.GetMatrix(), false );

// Create an Anasazi MultiVector, based on Epetra MultiVector
const Epetra_Map * Map = &(A->RowMap());
Teuchos::RCP<Epetra_MultiVector> ivec =
  Teuchos::rcp( new Epetra_MultiVector(*Map,blocksize) );

// Fill it with random numbers
ivec->Random();

// Setup the eigenproblem, with the matrix A and the initial vectors ivec
Teuchos::RCP< Anasazi::BasicEigenproblem<double,MV,OP> > MyProblem =
  Teuchos::rcp( new Anasazi::BasicEigenproblem<double,MV,OP>(A, ivec) );

// The problem is non-symmetric. Specify this in the eigenproblem.
MyProblem->setHermitian(false);

// Specify the desired number of eigenvalues
MyProblem->setNEV( nev );

// Signal that we are done setting up the eigenvalue problem
ierr = MyProblem->setProblem();

// Check the return from setProblem(). If this is true, there was an
```

```
  // error. This probably means we did not specify enough information for
  // the eigenproblem.
  assert(ierr);

  // Specify the verbosity level. Options include:
  // Anasazi::Errors
  //   This option is always set
  // Anasazi::Warnings
  //   Warnings (less severe than errors)
  // Anasazi::IterationDetails
  //   Details at each iteration, such as the current eigenvalues
  // Anasazi::OrthoDetails
  //   Details about orthogonality
  // Anasazi::TimingDetails
  //   A summary of the timing info for the solve() routine
  // Anasazi::FinalSummary
  //   A final summary
  // Anasazi::Debug
  //   Debugging information
  int verbosity = Anasazi::Warnings + Anasazi::Errors + Anasazi::FinalSummary;

  // Choose which eigenvalues to compute
  // Choices are:
  // LM - target the largest magnitude  [default]
  // SM - target the smallest magnitude
  // LR - target the largest real
  // SR - target the smallest real
  // LI - target the largest imaginary
  // SI - target the smallest imaginary
  std::string which("SM");

  // Create the parameter list for the eigensolver
  Teuchos::ParameterList MyPL;
  MyPL.set( "Verbosity", verbosity );
  MyPL.set( "Which", which );
  MyPL.set( "Block Size", blocksize );
  MyPL.set( "Num Blocks", 5 );
  MyPL.set( "Maximum Restarts", 300 );
  MyPL.set( "Convergence Tolerance", 1.0e-8 );

  // Create the Block Krylov Schur solver
  // This takes as inputs the eigenvalue problem and the solver parameters
  Anasazi::BlockKrylovSchurSolMgr<double,MV,OP>
    MyBlockKrylovSchur(MyProblem, MyPL );

  // Solve the eigenvalue problem, and save the return code
  Anasazi::ReturnType solverreturn = MyBlockKrylovSchur.solve();

  // Check return code of the solver: Unconverged, Failed, or OK
  switch (solverreturn) {

  // UNCONVERGED
  case Anasazi::Unconverged:
    if (verbose)
      cout << "Anasazi::BlockKrylovSchur::solve() did not converge!" << endl;
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    return 0;
    break;

  // CONVERGED
```

```
case Anasazi::Converged:
  if (verbose)
    cout << "Anasazi::BlockKrylovSchur::solve() converged!" << endl;
  break;
}

// Get eigensolution struct
Anasazi::Eigensolution<double, Epetra_MultiVector> sol = MyProblem->getSolution
    ();

// Get the number of eigenpairs returned
int numev = sol.numVecs;

// Get the index vector for the eigenvector storage
std::vector<int> index = sol.index;

// Get eigenvectors
Teuchos::RCP<Epetra_MultiVector> evecs = sol.Evecs;

// Get eigenvalues
// Because the matrix is nonsymmetric (and the eigenvalues are potentially
// complex), Evals is a vector of length numev where each entry is a pair
// with a real and imaginary part.
std::vector<Anasazi::Value<double> > evals = sol.Evals;

// Compute residuals.
Teuchos::LAPACK<int,double> lapack;
std::vector<double> normA(numev);

int i=0;
std::vector<int> curind(1);
std::vector<double> resnorm(1), tempnrm(1);
Teuchos::RCP<MV> evecr, eveci, tempAevec;
Epetra_MultiVector Aevec(*Map,numev);

// Compute A*evecs
OPT::Apply( *A, *evecs, Aevec );

Teuchos::SerialDenseMatrix<int,double> Breal(1,1), Bimag(1,1);
while (i<numev) {
  if (index[i]==0) {
    // Get a view of the current eigenvector (evecr)
    curind[0] = i;
    evecr = MVT::CloneView( *evecs, curind );

    // Get a copy of A*evecr
    tempAevec = MVT::CloneCopy( Aevec, curind );

    // Compute A*evecr - lambda*evecr
    Breal(0,0) = evals[i].realpart;
    MVT::MvTimesMatAddMv( -1.0, *evecr, Breal, 1.0, *tempAevec );

    // Compute the norm of the residual and increment counter
    MVT::MvNorm( *tempAevec, resnorm );
    normA[i] = resnorm[0]/Teuchos::ScalarTraits<double>::magnitude( evals[i].re
    alpart );
    i++;
  } else {
    // Get a view of the real part of the eigenvector (evecr)
    curind[0] = i;
    evecr = MVT::CloneView( *evecs, curind );
```

```
      // Get a copy of A*evecr
      tempAevec = MVT::CloneCopy( Aevec, curind );

      // Get a view of the imaginary part of the eigenvector (eveci)
      curind[0] = i+1;
      eveci = MVT::CloneView( *evecs, curind );

      // Set the eigenvalue into Breal and Bimag
      Breal(0,0) = evals[i].realpart;
      Bimag(0,0) = evals[i].imagpart;

      // Compute A*evecr - evecr*lambdar + eveci*lambdai
      MVT::MvTimesMatAddMv( -1.0, *evecr, Breal, 1.0, *tempAevec );
      MVT::MvTimesMatAddMv( 1.0, *eveci, Bimag, 1.0, *tempAevec );
      MVT::MvNorm( *tempAevec, tempnrm );

      // Get a copy of A*eveci
      tempAevec = MVT::CloneCopy( Aevec, curind );

      // Compute A*eveci - eveci*lambdar - evecr*lambdai
      MVT::MvTimesMatAddMv( -1.0, *evecr, Bimag, 1.0, *tempAevec );
      MVT::MvTimesMatAddMv( -1.0, *eveci, Breal, 1.0, *tempAevec );
      MVT::MvNorm( *tempAevec, resnorm );

      // Compute the norms and scale by magnitude of eigenvalue
      normA[i] = lapack.LAPY2( tempnrm[i], resnorm[i] ) /
  lapack.LAPY2( evals[i].realpart, evals[i].imagpart );
      normA[i+1] = normA[i];

      i=i+2;
    }
  }

  // Output results to screen
  if(verbose) {
    cout << scientific << showpos << setprecision(6) << showpoint;
    cout << "*****************************************************\n"
         << "           Results (outside of eigensolver)           \n"
         << "-----------------------------------------------------\n"
         << "  real(evals)\t\timag(evals)\tnormR\n"
         << "-----------------------------------------------------\n";
    for(int i=0; i<numev; ++i) {
      cout << "  " << evals[i].realpart << "\t\t" << evals[i].imagpart
           << "\t" << normA[i] << endl;
    }
    cout << "*****************************************************\n" << endl;
  }


#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
```

```
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
#endif

  puts("Please configure Didasko with:");
  puts("--enable-epetra");
  puts("--enable-teuchos");
  puts("--enable-triutils");
  puts("--enable-anasazi");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 6

anasazi_ex4

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Compute the smallest magnitude eigenvalues and the corresponding eigenvectors
      using
// Anasazi::LOBPCGSolMgr

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_ANASAZI) && defined(HAVE
      _DIDASKO_TEUCHOS) && defined(HAVE_DIDASKO_TRIUTILS)

#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif

#include "Epetra_MultiVector.h"
#include "Trilinos_Util_CrsMatrixGallery.h"
#include "AnasaziBasicEigenproblem.hpp"
#include "AnasaziEpetraAdapter.hpp"
#include "AnasaziLOBPCGSolMgr.hpp"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  // Initialize MPI and setup an Epetra communicator
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  // If we aren't using MPI, then setup a serial communicator.
  Epetra_SerialComm Comm;
#endif

  // Some typedefs for oft-used data types
```

```
typedef Epetra_MultiVector MV;
typedef Epetra_Operator OP;
typedef Anasazi::MultiVecTraits<double, Epetra_MultiVector> MVT;

bool ierr;

// Get our processor ID (0 if running serial)
int MyPID = Comm.MyPID();

// Verbose flag: only processor 0 should print
bool verbose = (MyPID==0);

// Initialize a Gallery object, from which we will select a matrix
// Select a 2-D laplacian, of order 100
Trilinos_Util::CrsMatrixGallery Gallery("laplace_2d", Comm);
Gallery.Set("problem_size", 100);

// Say hello and print some information about the gallery matrix
if (verbose) {
  cout << "Anasazi Example: LOBPCG" << endl;
  cout << "Problem info:" << endl;
  cout << Gallery;
  cout << endl;
}


// Setup some more problem/solver parameters:

// Number of desired eigenvalues
int nev = 4;

// Block size
int blocksize = 8;


// Get a pointer to the system matrix, inside of a Teuchos::RCP
// The Teuchos::RCP is a reference counting pointer that handles
// garbage collection for us, so that we can perform memory allocation without
// having to worry about freeing memory manually.
Teuchos::RCP<Epetra_CrsMatrix> A = Teuchos::rcp( Gallery.GetMatrix(), false );


// Create an Anasazi MultiVector, based on Epetra MultiVector
const Epetra_Map * Map = &(A->RowMap());
Teuchos::RCP<Epetra_MultiVector> ivec =
  Teuchos::rcp( new Epetra_MultiVector(*Map,blocksize) );

// Fill it with random numbers
ivec->Random();


// Setup the eigenproblem, with the matrix A and the initial vectors ivec
Teuchos::RCP< Anasazi::BasicEigenproblem<double,MV,OP> > MyProblem =
  Teuchos::rcp( new Anasazi::BasicEigenproblem<double,MV,OP>(A, ivec) );

// The 2-D laplacian is symmetric. Specify this in the eigenproblem.
MyProblem->setHermitian(true);

// Specify the desired number of eigenvalues
MyProblem->setNEV( nev );

// Signal that we are done setting up the eigenvalue problem
```

```
  ierr = MyProblem->setProblem();

  // Check the return from setProblem(). If this is true, there was an
  // error. This probably means we did not specify enough information for
  // the eigenproblem.
  assert(ierr);

  // Specify the verbosity level. Options include:
  // Anasazi::Errors
  //   This option is always set
  // Anasazi::Warnings
  //   Warnings (less severe than errors)
  // Anasazi::IterationDetails
  //   Details at each iteration, such as the current eigenvalues
  // Anasazi::OrthoDetails
  //   Details about orthogonality
  // Anasazi::TimingDetails
  //   A summary of the timing info for the solve() routine.
  // Anasazi::FinalSummary
  //   A final summary
  // Anasazi::Debug
  //   Debugging information
  int verbosity = Anasazi::Warnings + Anasazi::Errors + Anasazi::FinalSummary;

  // Choose which eigenvalues to compute
  // Choices are:
  // LM - target the largest magnitude
  // SM - target the smallest magnitude
  // LR - target the largest real
  // SR - target the smallest real
  std::string which("SM");

  // Create the parameter list for the eigensolver
  Teuchos::ParameterList MyPL;
  MyPL.set( "Verbosity", verbosity );
  MyPL.set( "Which", which );
  MyPL.set( "Block Size", blocksize );
  MyPL.set( "Maximum Iterations", 500 );
  MyPL.set( "Convergence Tolerance", 1.0e-8 );

  // Create the LOBPCG solver
  // This takes as inputs the eigenvalue problem that we constructed,
  // the output manager, and the solver parameters
  Anasazi::LOBPCGSolMgr<double,MV,OP> MyLOBPCG(MyProblem, MyPL );

  // Solve the eigenvalue problem, and save the return code
  Anasazi::ReturnType solverreturn = MyLOBPCG.solve();

  // Check return code of the solver: Unconverged, Failed, or OK
  switch (solverreturn) {

  // UNCONVERGED
  case Anasazi::Unconverged:
    if (verbose)
      cout << "Anasazi::LOBPCG::solve() did not converge!" << endl;
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    return 0;
    break;

  // CONVERGED
```

```
  case Anasazi::Converged:
    if (verbose)
      cout << "Anasazi::LOBPCG::solve() converged!" << endl;
    break;
  }

  // Get eigensolution struct
  Anasazi::Eigensolution<double, Epetra_MultiVector> sol = MyProblem->getSolution
      ();

  // Get the number of eigenpairs returned
  int numev = sol.numVecs;

  // Get eigenvectors
  Teuchos::RCP<Epetra_MultiVector> evecs = sol.Evecs;

  // Get eigenvalues
  std::vector<Anasazi::Value<double> > evals = sol.Evals;

  // Test residuals
  // Generate a (numev x numev) dense matrix for the eigenvalues...
  // This matrix is automatically initialized to zero
  Teuchos::SerialDenseMatrix<int, double> D(numev, numev);

  // Add the eigenvalues on the diagonals
  for (int i=0; i<numev; i++) {
    D(i,i) = evals[i].realpart;
  }

  // Generate a multivector for the product of the matrix and the eigenvectors
  Epetra_MultiVector R(*Map, numev);

  // R = A*evecs
  A->Apply( *evecs, R );

  // R -= evecs*D
  //    = A*evecs - evecs*D
  MVT::MvTimesMatAddMv( -1.0, *evecs, D, 1.0, R );

  // Compute the 2-norm of each vector in the MultiVector
  // and store them to a std::vector<double>
  std::vector<double> normR(numev);
  MVT::MvNorm( R, normR );

  // Output results to screen
  if(verbose) {
    cout << scientific << showpos << setprecision(6) << showpoint;
    cout << "*******************************************************\n"
         << "           Results (outside of eigensolver)           \n"
         << "-------------------------------------------------------\n"
         << "  evals\t\t\tnormR\n"
         << "-------------------------------------------------------\n";
    for( int i=0 ; i<numev ; ++i ) {
      cout << "  " << evals[i].realpart << "\t\t" << normR[i] << endl;
    }
    cout << "*******************************************************\n" << endl;
  }

  // Perform MPI cleanup, if we are using MPI
#ifdef HAVE_MPI
  MPI_Finalize();
#endif
```

```
  return(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
#endif

  puts("Please configure Didasko with:");
  puts("--enable-epetra");
  puts("--enable-teuchos");
  puts("--enable-triutils");
  puts("--enable-anasazi");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 7

aztecoo_ex1

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Solve a 2D Laplacian problem
// This example builds the matrix and solves it with AztecOO.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_AZTECOO)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"
#include "AztecOO.h"

// external function
void  get_neighbours( const int i, const int nx, const int ny,
          int & left, int & right,
          int & lower, int & upper);

// =========== //
// main driver //
// =========== //

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
```

```
#else
  Epetra_SerialComm Comm;
#endif

  // number of nodes along the x- and y-axis
  int nx = 5;
  int ny = 6;
  int NumGlobalElements = nx * ny;

  // create a linear map
  Epetra_Map Map(NumGlobalElements,0,Comm);

  // local number of rows
  int NumMyElements = Map.NumMyElements();
  // get update list
  int * MyGlobalElements = new int [NumMyElements];
  Map.MyGlobalElements( MyGlobalElements );

  // Create a Epetra_Matrix with 5 nonzero per rows

  Epetra_CrsMatrix A(Copy,Map,5);

  // Add  rows one-at-a-time
  // Need some vectors to help

  double Values[4];
  int Indices[4];
  int NumEntries;
  int left, right, lower, upper;
  double diag = 4.0;

  for( int i=0 ; i<NumMyElements; ++i ) {
    int NumEntries=0;
    get_neighbours(  MyGlobalElements[i], nx, ny,
         left, right, lower, upper);
    if( left != -1 ) {
  Indices[NumEntries] = left;
  Values[NumEntries] = -1.0;
  ++NumEntries;
    }
    if( right != -1 ) {
      Indices[NumEntries] = right;
      Values[NumEntries] = -1.0;
      ++NumEntries;
    }
    if( lower != -1 ) {
      Indices[NumEntries] = lower;
      Values[NumEntries] = -1.0;
      ++NumEntries;
    }
    if( upper != -1 ) {
      Indices[NumEntries] = upper;
      Values[NumEntries] = -1.0;
      ++NumEntries;
    }
    // put the off-diagonal entries
    A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
    // Put in the diagonal entry
    A.InsertGlobalValues(MyGlobalElements[i], 1, &diag, MyGlobalElements+i);
  }

  // Finish up
```

```
  A.FillComplete();

  // create x and b vectors
  Epetra_Vector x(Map);
  Epetra_Vector b(Map);
  b.PutScalar(1.0);

  // ================== AZTECOO INTERFACE ======================

  // create linear problem
  Epetra_LinearProblem Problem(&A,&x,&b);
  // create AztecOO instance
  AztecOO Solver(Problem);

  Solver.SetAztecOption( AZ_precond, AZ_Jacobi );

  Solver.Iterate(1000,1E-9);

  // ================== END OF AZTECOO INTERFACE ================

  if( Comm.MyPID() == 0 ) {
    cout << "Solver performed " << Solver.NumIters()
   << "iterations.\n";
    cout << "Norm of the true residual = " << Solver.TrueResidual() << endl;
  }

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

}

/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/

void  get_neighbours( const int i, const int nx, const int ny,
          int & left, int & right,
          int & lower, int & upper)
{

  int ix, iy;
  ix = i%nx;
  iy = (i - ix)/nx;

  if( ix == 0 )
    left = -1;
  else
    left = i-1;
  if( ix == nx-1 )
    right = -1;
  else
    right = i+1;
  if( iy == 0 )
    lower = -1;
  else
    lower = i-nx;
  if( iy == ny-1 )
    upper = -1;
  else
```

```
    upper = i+nx;

  return;

}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-aztecoo");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return 0;
}

#endif
```

Chapter 8

aztecoo_ex2

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

//  Solve a linear system with Aztec, using Aztec as a preconditioner
// (recursive way)
//
// NOTE: This example first builds the matrix, then solves it with AztecOO
//
// NOTE2: this example implemenets minor modifications to one of the
// examples included in the AztecOO package. Please give a look  to file
// ${TRILINOS_HOME}/packages/aztecoo/examples/AztecOO_RecursiveCall/cxx_main.cpp
// for more details.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_TRIUTILS)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "Epetra_MpiComm.h"
#include "mpi.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_LinearProblem.h"
#include "AztecOO.h"
#include "AztecOO_Operator.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
```

```
#else
  Epetra_SerialComm Comm;
#endif

  /* this example creates a tridiagonal matrix of type
   *
   *     |  2  -1            |
   *     | -1   2   -1       |
   * A = |      ...  ... ... |
   *     |            -1  2  |
   */

  // set global dimension to 5, could be any number
  int NumGlobalElements = 5;

  // create a linear map
  Epetra_Map Map(NumGlobalElements,0,Comm);
  // local number of rows
  int NumMyElements = Map.NumMyElements();
  // get update list
  int * MyGlobalElements = new int [NumMyElements];
  Map.MyGlobalElements( MyGlobalElements );

  // Create a Epetra_Matrix
  // create a CSR matrix

  Epetra_CrsMatrix A(Copy,Map,3);

  // Add  rows one-at-a-time
  // Need some vectors to help
  // Off diagonal Values will always be -1

  double *Values = new double[2];
  Values[0] = -1.0; Values[1] = -1.0;
  int *Indices = new int[2];
  double two = 2.0;
  int NumEntries;

  for( int i=0 ; i<NumMyElements; ++i ) {
    if (MyGlobalElements[i]==0) {
  Indices[0] = 1;
  NumEntries = 1;
    } else if (MyGlobalElements[i] == NumGlobalElements-1) {
      Indices[0] = NumGlobalElements-2;
      NumEntries = 1;
    } else {
      Indices[0] = MyGlobalElements[i]-1;
      Indices[1] = MyGlobalElements[i]+1;
      NumEntries = 2;
    }
    A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
    // Put in the diagonal entry
    A.InsertGlobalValues(MyGlobalElements[i], 1, &two, MyGlobalElements+i);
  }

  // Finish up
  A.FillComplete();

  // E N D   O F   M A T R I X   C O N S T R U C T I O N

  Epetra_Vector x(Map);
  Epetra_Vector b(Map);
```

```
  x.Random();

  // this is the linear problem to be solved: set the linear operator,
  // the solution and the right-hand side
  Epetra_LinearProblem A_Problem(&A, &x, &b);

  // and this is the AztecOO solver
  AztecOO A_Solver(A_Problem);

  // --- Here we define the precondioner ---
  // create P as a copy of A, in principle can be different
  Epetra_CrsMatrix P(A);

  // Here we create the linear problem which will be used as
  // preconditioner. This requires sereval steps.
  // (Note that all the P_ prefix indentify preconditioner'
  // objects)

  // 1. we create the linear system solve at each prec step
  Epetra_LinearProblem P_Problem;
  // and we assign the linear operator (in this case, the
  // matrix A itself)
  P_Problem.SetOperator(&P);

  // as we wish to use AztecOO to solve the prec step
  // (in a recursive way), we have to define an AztecOO
  // object.
  AztecOO P_Solver(P_Problem);

  // now, we customize certain parameters
  P_Solver.SetAztecOption(AZ_precond, AZ_Jacobi);
  P_Solver.SetAztecOption(AZ_output, AZ_none);
  P_Solver.SetAztecOption(AZ_solver, AZ_cg);
  // The last step is to create an AztecOO_Operator, so that
  // we can set the Aztec's preconditioner with.
  AztecOO_Operator P_Operator(&P_Solver, 10);

  // Here we set the user's defined preconditioners
  A_Solver.SetPrecOperator(&P_Operator);

  // --- up to here ---

  // Finally, we solve the linear system:

  int Niters=100;
  A_Solver.SetAztecOption(AZ_kspace, Niters);
  A_Solver.SetAztecOption(AZ_solver, AZ_GMRESR);

  A_Solver.Iterate(Niters, 1.0E-12);

#ifdef HAVE_MPI
  MPI_Finalize() ;
#endif

  return 0;

}

#else

#include <stdlib.h>
```

```
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-triutils\n"
       "--enable-aztecoo\n");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return 0;
}

#endif
```

Chapter 9

aztecoo_ex3

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Solve a linear system with AztecOO.
// The linear system is created using MatrixGallery

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_TRIUTILS)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "AztecOO.h"
#include "Epetra_CrsMatrix.h"

#include "Trilinos_Util_CrsMatrixGallery.h"

using namespace Trilinos_Util;

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // initialize an Gallery object
  CrsMatrixGallery Gallery("laplace_2d", Comm);
  Gallery.Set("problem_size",900);
```

```
  // Get pointers to the linear problem (containing matrix, LHS and RHS).
  Epetra_LinearProblem * Problem= Gallery.GetLinearProblem();

  // initialize the AztecOO solve object, based on current linear problem
  AztecOO solver(*Problem);

  // here set some AztecOO options:
  // - symmetric problem;
  // - domain decomposition preconditioner
  // - ICC factorization on each subdomain
  solver.SetAztecOption(AZ_solver, AZ_cg_condnum);
  solver.SetAztecOption(AZ_precond, AZ_dom_decomp);
  solver.SetAztecOption(AZ_overlap,0);
  solver.SetAztecOption(AZ_subdomain_solve, AZ_icc);

  // solve the linear system
  solver.Iterate(1550, 1e-12);

  // AztecOO defined a certain number of output parameters, and store them
  // in a double vector called status.
  double status[AZ_STATUS_SIZE];
  solver.GetAllAztecStatus(status);

  // verify that linear system has been solved as required
  double residual, diff;

  Gallery.ComputeResidual(&residual);
  Gallery.ComputeDiffBetweenStartingAndExactSolutions(&diff);

  if( Comm.MyPID()==0 ) {
    cout << "||b-Ax||_2 = " << residual << endl;
    cout << "||x_exact - x||_2 = " << diff << endl;
  }

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-triutils\n"
       "--enable-aztecoo\n");
```

```
#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 10

ifpack_ex1

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Using IFPACK factorizations as Aztec's preconditioners
//
// NOTE: this example implemenets minor modifications to one of the
// examples included in the AztecOO package. Please give a look
// to file ${TRILINOS_HOME}/packages/aztecoo/examples/IfpackAztecOO/cxx_main.cpp
// for more details.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_IFPACK) && defined(HAVE_
      DIDASKO_AZTECOO)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Comm.h"
#include "Epetra_Map.h"
#include "Epetra_Time.h"
#include "Epetra_BlockMap.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Vector.h"
#include "Epetra_Export.h"
#include "AztecOO.h"
#include "Trilinos_Util.h"
#include "Epetra_VbrMatrix.h"
#include "Epetra_CrsMatrix.h"
#include "Ifpack_CrsIct.h"

// function for fancy output
```

```
string toString(const int& x) {
  char s[100];
  sprintf(s, "%d", x);
  return string(s);
}

string toString(const double& x) {
  char s[100];
  sprintf(s, "%g", x);
  return string(s);
}

// main driver

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm (MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  int MyPID = Comm.MyPID();
  bool verbose = false;
  if (MyPID==0) verbose = true;

  // matrix downloaded from MatrixMarket
  char FileName[] = "../HBMatrices/fidap005.rua";

  Epetra_Map * readMap; // Pointers because of Trilinos_Util_ReadHb2Epetra
  Epetra_CrsMatrix * readA;
  Epetra_Vector * readx;
  Epetra_Vector * readb;
  Epetra_Vector * readxexact;

  // Call routine to read in HB problem
  Trilinos_Util_ReadHb2Epetra(FileName, Comm, readMap, readA, readx,
            readb, readxexact);

  int NumGlobalElements = readMap->NumGlobalElements();

  // Create uniform distributed map
  Epetra_Map map(NumGlobalElements, 0, Comm);

  // Create Exporter to distribute read-in matrix and vectors

  Epetra_Export exporter(*readMap, map);
  Epetra_CrsMatrix A(Copy, map, 0);
  Epetra_Vector x(map);
  Epetra_Vector b(map);
  Epetra_Vector xexact(map);

  Epetra_Time FillTimer(Comm);
  A.Export(*readA, exporter, Add);
  x.Export(*readx, exporter, Add);
  b.Export(*readb, exporter, Add);
  xexact.Export(*readxexact, exporter, Add);

  A.FillComplete();

  delete readA;
```

```
  delete readx;
  delete readb;
  delete readxexact;
  delete readMap;

  // ============================ //
  // Construct ILU preconditioner //
  // ---------------------------- //

  //  modify those parameters
  int    LevelFill = 1;
  double DropTol = 0.0;
  double Condest;

  Ifpack_CrsIct * ICT = NULL;
  ICT = new Ifpack_CrsIct(A,DropTol,LevelFill);
  // Init values from A
  ICT->InitValues(A);
  // compute the factors
  ICT->Factor();
  // and now estimate the condition number
  ICT->Condest(false,Condest);

  cout << Condest << endl;

  if( Comm.MyPID() == 0 ) {
    cout << "Condition number estimate (level-of-fill = "
   << LevelFill <<  ") = " << Condest << endl;
  }

  // Define label for printing out during the solve phase
  string label = "Ifpack_CrsIct Preconditioner: LevelFill = " + toString(LevelFil
      l) +
                                                 " Overlap = 0";
  ICT->SetLabel(label.c_str());

  // Here we create an AztecOO object
  AztecOO solver;
  solver.SetUserMatrix(&A);
  solver.SetLHS(&x);
  solver.SetRHS(&b);
  solver.SetAztecOption(AZ_solver,AZ_cg);

  // Here we set the IFPACK preconditioner and specify few parameters

  solver.SetPrecOperator(ICT);

  int Niters = 1200;
  solver.SetAztecOption(AZ_kspace, Niters);
  solver.SetAztecOption(AZ_output, 20);
  solver.Iterate(Niters, 5.0e-5);

  if (ICT!=0) delete ICT;

#ifdef HAVE_MPI
  MPI_Finalize() ;
#endif

return 0 ;
}

#else
```

```
#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:");
  puts("--enable-epetra");
  puts("--enable-ifpack");
  puts("--enable-aztecoo");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 11

ifpack_ex2

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Using IFPACK factorizations as Aztec's preconditioners

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_IFPACK) && defined(HAVE_
      DIDASKO_AZTECOO)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Trilinos_Util.h"
#include "Epetra_Comm.h"
#include "Epetra_Map.h"
#include "Epetra_Time.h"
#include "Epetra_BlockMap.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Vector.h"
#include "AztecOO.h"
#include "Epetra_Export.h"
#include "Epetra_CrsMatrix.h"
#include "Ifpack_IlukGraph.h"
#include "Ifpack_CrsRiluk.h"

// function for fancy output

string toString(const int& x) {
  char s[100];
  sprintf(s, "%d", x);
  return string(s);
}
```

```
string toString(const double& x) {
  char s[100];
  sprintf(s, "%g", x);
  return string(s);
}

// main driver

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm (MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  int MyPID = Comm.MyPID();
  bool verbose = false;
  if (MyPID==0) verbose = true;

  // B E G I N   O F   M A T R I X   C O N S T R U C T I O N

  // matrix downloaded from MatrixMarket
  char FileName[] = "../HBMatrices/fidap005.rua";

  Epetra_Map * readMap; // Pointers because of Trilinos_Util_ReadHb2Epetra
  Epetra_CrsMatrix * readA;
  Epetra_Vector * readx;
  Epetra_Vector * readb;
  Epetra_Vector * readxexact;

  // Call routine to read in HB problem
  Trilinos_Util_ReadHb2Epetra(FileName, Comm, readMap, readA, readx,
            readb, readxexact);

  int NumGlobalElements = readMap->NumGlobalElements();

  // Create uniform distributed map
  Epetra_Map map(NumGlobalElements, 0, Comm);

  // Create Exporter to distribute read-in matrix and vectors

  Epetra_Export exporter(*readMap, map);
  Epetra_CrsMatrix A(Copy, map, 0);
  Epetra_Vector x(map);
  Epetra_Vector b(map);
  Epetra_Vector xexact(map);

  Epetra_Time FillTimer(Comm);
  A.Export(*readA, exporter, Add);
  x.Export(*readx, exporter, Add);
  b.Export(*readb, exporter, Add);
  xexact.Export(*readxexact, exporter, Add);

  A.FillComplete();

  delete readA;
  delete readx;
  delete readb;
  delete readxexact;
```

```
  delete readMap;

  // E N D   O F   M A T R I X   C O N S T R U C T I O N

  // ============================== //
  // Construct RILU preconditioner //
  // ----------------=------------- //

  //  modify those parameters
  int    LevelFill = 0;
  int    Overlap = 2;
  double Athresh = 0.0;
  double Rthresh = 1.0;

  Ifpack_IlukGraph * Graph = 0;
  Ifpack_CrsRiluk * RILU = 0;

  Graph = new Ifpack_IlukGraph(A.Graph(), LevelFill, Overlap);
  Graph->ConstructFilledGraph();

  RILU = new Ifpack_CrsRiluk(*Graph);
  int initerr = RILU->InitValues(A);
  if (initerr!=0) cout << Comm << "*ERR* InitValues = " << initerr;

  RILU->Factor();

  // Define label for printing out during the solve phase
  string label = "Ifpack_CrsRiluk Preconditioner: LevelFill = " + toString(LevelF
      ill) +
                                                 " Overlap = " + toString(Overlap
      ) +
                                                 " Athresh = " + toString(Athresh
      ) +
                                                 " Rthresh = " + toString(Rthresh
      );
  RILU->SetLabel(label.c_str());

  // Here we create an AztecOO object
  AztecOO solver;
  solver.SetUserMatrix(&A);
  solver.SetLHS(&x);
  solver.SetRHS(&b);

  // Here we set the IFPACK preconditioner and specify few parameters

  solver.SetPrecOperator(RILU);

  int Niters = 1200;
  solver.SetAztecOption(AZ_kspace, Niters);
  solver.Iterate(Niters, 5.0e-6);

  if (RILU!=0) delete RILU;
  if (Graph!=0) delete Graph;

#ifdef HAVE_MPI
  MPI_Finalize() ;
#endif

return 0 ;
}

#else
```

```
#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-ifpack\n"
       "--enable-aztecoo");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 12

nox_ex1

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Trilinos Tutorial
// -----------------
// Simple nonlinear problem.
// This file shows how to solve the nonlinear problem
//
// x(0)^2 + x(1)^2 -1 = 0
//      x(1) - x(0)^2 = 0
//
// using NOX. Due to the very small dimension of the problem,
// it should be run with one process.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_NOX) && defined(HAVE_NOX
      _EPETRA)

#include <iostream>
#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_RowMatrix.h"
#include "Epetra_CrsMatrix.h"
#include "NOX.H"
#include "NOX_Epetra_Interface_Required.H"
#include "NOX_Epetra_Interface_Jacobian.H"
#include "NOX_Epetra_LinearSystem_AztecOO.H"
#include "NOX_Epetra_Group.H"
```

```
class SimpleProblemInterface : public NOX::Epetra::Interface::Required,
                               public NOX::Epetra::Interface::Jacobian
{

public:

  SimpleProblemInterface( Epetra_Vector & InitialGuess,
                          Epetra_Vector & ExactSolution )
  {
    InitialGuess_ = new Epetra_Vector(InitialGuess);
    ExactSolution_ = new Epetra_Vector(ExactSolution);
  };

  ~SimpleProblemInterface()
  {
  };

  bool computeF(const Epetra_Vector & x, Epetra_Vector & f,
                NOX::Epetra::Interface::Required::FillType F )
  {
    f[0] = x[0]*x[0] + x[1]*x[1] - 1.0;
    f[1] = x[1] - x[0]*x[0];
    return true;
  };

  bool computeJacobian(const Epetra_Vector & x, Epetra_Operator & Jac)
  {

    Epetra_CrsMatrix * J;
    J = dynamic_cast<Epetra_CrsMatrix*>(&Jac);
    if (J == NULL) {
      cout << "*ERR* Problem_Interface::computeJacobian() - The supplied" << endl
      ;
      cout << "*ERR* Epetra_Operator is NOT an Epetra_CrsMatrix!" << endl;
      throw;
    }

    std::vector<int> indices(2);
    std::vector<double> values(2);

    indices[0] = 0;
    indices[1] = 1;

    // Row 0
    values[0] = 2.0 * x[0];
    values[1] = 2.0 * x[1];
    J->ReplaceGlobalValues(0, 2, &values[0], &indices[0]);

    // Row 1
    values[0] = - 2.0 * x[0];
    values[1] = 1.0;
    J->ReplaceGlobalValues(1, 2, &values[0], &indices[0]);

    return true;
  }

  bool computePrecMatrix(const Epetra_Vector & x, Epetra_RowMatrix & M)
  {
    cout << "*ERR* SimpleProblem::preconditionVector()\n";
    cout << "*ERR* don't use explicit preconditioning" << endl;
    exit( 0 );
    throw 1;
```

```
  }

  bool computePreconditioner(const Epetra_Vector & x, Epetra_Operator & O)
  {
    cout << "*ERR* SimpleProblem::preconditionVector()\n";
    cout << "*ERR* don't use explicit preconditioning" << endl;
    exit( 0 );
        throw 1;
  }

private:
  Epetra_Vector * InitialGuess_;
  Epetra_Vector * ExactSolution_;

};

// =========== //
// main driver //
// =========== //

int main( int argc, char **argv )
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  if (Comm.NumProc() != 1) {
    if (Comm.MyPID() == 0)
      cerr << "Please run this test with one process only!" << endl;
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    exit(EXIT_SUCCESS);
  }

  // linear map for the 2 global elements
  Epetra_Map Map(2,0,Comm);

  // build up initial guess and exact solution
  Epetra_Vector ExactSolution(Map);
  ExactSolution[0] = sqrt(0.5*(sqrt(5.0)-1));
  ExactSolution[1] = 0.5*(sqrt(5.0)-1);

  Epetra_Vector InitialGuess(Map);
  InitialGuess[0] = 0.5;
  InitialGuess[1] = 0.5;

  // Set up the problem interface
  Teuchos::RCP<SimpleProblemInterface> interface =
    Teuchos::rcp(new SimpleProblemInterface(InitialGuess,ExactSolution) );

  // Create the top level parameter list
  Teuchos::RCP<Teuchos::ParameterList> nlParamsPtr =
    Teuchos::rcp(new Teuchos::ParameterList);
  Teuchos::ParameterList& nlParams = *(nlParamsPtr.get());

  // Set the nonlinear solver method
  nlParams.set("Nonlinear Solver", "Line Search Based");
```

```
// Set the printing parameters in the "Printing" sublist
Teuchos::ParameterList& printParams = nlParams.sublist("Printing");
printParams.set("MyPID", Comm.MyPID());
printParams.set("Output Precision", 3);
printParams.set("Output Processor", 0);
printParams.set("Output Information",
    NOX::Utils::OuterIteration +
    NOX::Utils::OuterIterationStatusTest +
    NOX::Utils::InnerIteration +
    NOX::Utils::Parameters +
    NOX::Utils::Details +
    NOX::Utils::Warning);

// start definition of nonlinear solver parameters
// Sublist for line search
Teuchos::ParameterList& searchParams = nlParams.sublist("Line Search");
searchParams.set("Method", "Full Step");

// Sublist for direction
Teuchos::ParameterList& dirParams = nlParams.sublist("Direction");
dirParams.set("Method", "Newton");

Teuchos::ParameterList& newtonParams = dirParams.sublist("Newton");
newtonParams.set("Forcing Term Method", "Constant");

// Sublist for linear solver for the Newton method
Teuchos::ParameterList& lsParams = newtonParams.sublist("Linear Solver");
lsParams.set("Aztec Solver", "GMRES");
lsParams.set("Max Iterations", 800);
lsParams.set("Tolerance", 1e-4);
lsParams.set("Output Frequency", 50);
lsParams.set("Aztec Preconditioner", "ilu");

// Build the Jacobian matrix
Teuchos::RCP<Epetra_CrsMatrix> A =
  Teuchos::rcp(new Epetra_CrsMatrix(Copy,Map,2));
{
  std::vector<int> indices(2);
  std::vector<double> values(2);
  indices[0]=0;
  indices[1]=1;

  values[0] = 2.0 * InitialGuess[0];
  values[1] = 2.0 * InitialGuess[1];
  A.get()->InsertGlobalValues(0, 2, &values[0], &indices[0]);
  values[0] = - 2.0 * InitialGuess[0];
  values[1] = 1.0;
  A.get()->InsertGlobalValues(1, 2, &values[0], &indices[0]);

  A.get()->FillComplete();
}

Teuchos::RCP<NOX::Epetra::Interface::Required> iReq = interface;
Teuchos::RCP<NOX::Epetra::Interface::Jacobian> iJac = interface;
Teuchos::RCP<NOX::Epetra::LinearSystemAztecOO> linSys =
  Teuchos::rcp(new NOX::Epetra::LinearSystemAztecOO(printParams, lsParams,
               iReq,
               iJac, A,
               InitialGuess));

// Need a NOX::Epetra::Vector for constructor
```

```
  NOX::Epetra::Vector noxInitGuess(InitialGuess, NOX::DeepCopy);
  Teuchos::RCP<NOX::Epetra::Group> grpPtr =
    Teuchos::rcp(new NOX::Epetra::Group(printParams,
          iReq,
          noxInitGuess,
          linSys));

  // Set up the status tests
  Teuchos::RCP<NOX::StatusTest::NormF> testNormF =
    Teuchos::rcp(new NOX::StatusTest::NormF(1.0e-4));
  Teuchos::RCP<NOX::StatusTest::MaxIters> testMaxIters =
    Teuchos::rcp(new NOX::StatusTest::MaxIters(20));
  // this will be the convergence test to be used
  Teuchos::RCP<NOX::StatusTest::Combo> combo =
    Teuchos::rcp(new NOX::StatusTest::Combo(NOX::StatusTest::Combo::OR,
              testNormF, testMaxIters));

  // Create the solver
  Teuchos::RCP<NOX::Solver::Generic> solver =
    NOX::Solver::buildSolver(grpPtr, combo, nlParamsPtr);

  // Solve the nonlinesar system
  NOX::StatusTest::StatusType status = solver->solve();

  if( NOX::StatusTest::Converged  != status )
    cout << "\n" << "-- NOX solver converged --" << "\n";
  else
    cout << "\n" << "-- NOX solver did not converge --" << "\n";

  // Print the answer
  cout << "\n" << "-- Parameter List From Solver --" << "\n";
  solver->getList().print(cout);

  // Get the Epetra_Vector with the final solution from the solver
  const NOX::Epetra::Group & finalGroup =
    dynamic_cast<const NOX::Epetra::Group&>(solver->getSolutionGroup());
  const Epetra_Vector & finalSolution =
      (dynamic_cast<const NOX::Epetra::Vector&>(finalGroup.getX())).getEpetraVect
      or();

  if( Comm.MyPID() == 0 ) cout << "Computed solution : " << endl;
  cout << finalSolution;

  if( Comm.MyPID() == 0 ) cout << "Exact solution : " << endl;
  cout << ExactSolution;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
```

```
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif
  puts("Please configure Didasko with:");
  puts("--enable-epetra");
  puts("--enable-nox-epetra");
  puts("--enable-ifpack");
  puts("--enable-aztecoo");
  puts("--enable-nox");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return(EXIT_SUCCESS);
}
#endif
```

Chapter 13

nox_ex2

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Trilinos Tutorial
// -----------------
// Simple nonlinear PDE problem.
// This file shows how to solve the nonlinear problem
//
// -\Delta u + \lambda e^u = 0  in \Omega = (0,1) \times (0,1)
//                        u = 0  on \partial \Omega
//
// using NOX

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_NOX) && defined(HAVE_NOX
      _EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_RowMatrix.h"
#include "Epetra_CrsMatrix.h"
#include "NOX.H"
#include "NOX_Epetra_Interface_Required.H"
#include "NOX_Epetra_Interface_Jacobian.H"
#include "NOX_Epetra_LinearSystem_AztecOO.H"
#include "NOX_Epetra_Group.H"

// this is required to know the number of lower, upper, left and right
// node for each node of the Cartesian grid (composed by nx \timex ny
```

```
// elements)

static void  get_neighbours( const int i, const int nx, const int ny,
           int & left, int & right,
           int & lower, int & upper)
{

  int ix, iy;
  ix = i%nx;
  iy = (i - ix)/nx;

  if( ix == 0 )
    left = -1;
  else
    left = i-1;
  if( ix == nx-1 )
    right = -1;
  else
    right = i+1;
  if( iy == 0 )
    lower = -1;
  else
    lower = i-nx;
  if( iy == ny-1 )
    upper = -1;
  else
    upper = i+nx;

  return;

}

// This function creates a CrsMatrix, whose elements corresponds
// to the discretization of a Laplacian over a Cartesian grid,
// with nx grid point along the x-axis and and ny grid points
// along the y-axis. For the sake of simplicity, I suppose that
// all the nodes in the matrix are internal nodes (Dirichlet
// boundary nodes are supposed to have been already condensated)

Epetra_CrsMatrix * CreateLaplacian( const int nx, const int ny,
            const Epetra_Comm * Comm)
{

  int NumGlobalElements = nx * ny;

  // create a map
  Epetra_Map * Map = new Epetra_Map(NumGlobalElements,0,*Comm);
  // local number of rows
  int NumMyElements = Map->NumMyElements();
  // get update list
  int * MyGlobalElements = Map->MyGlobalElements();

  double hx = 1.0/(nx-1);
  double hy = 1.0/(ny-1);
  double off_left  = -1.0/(hx*hx);
  double off_right = -1.0/(hx*hx);
  double off_lower = -1.0/(hy*hy);
  double off_upper = -1.0/(hy*hy);
  double diag      =  2.0/(hx*hx) + 2.0/(hy*hy);

  int left, right, lower, upper;
```

```
  // a bit overestimated the nonzero per row

  Epetra_CrsMatrix * A = new Epetra_CrsMatrix(Copy,*Map,5);

  // Add  rows one-at-a-time

  double *Values = new double[4];
  int *Indices = new int[4];

  for( int i=0 ; i<NumMyElements; ++i ) {
    int NumEntries=0;
    get_neighbours(  MyGlobalElements[i], nx, ny,
         left, right, lower, upper);
    if( left != -1 ) {
      Indices[NumEntries] = left;
      Values[NumEntries] = off_left;
      ++NumEntries;
    }
    if( right != -1 ) {
      Indices[NumEntries] = right;
      Values[NumEntries] = off_right;
      ++NumEntries;
    }
    if( lower != -1 ) {
      Indices[NumEntries] = lower;
      Values[NumEntries] = off_lower;
      ++NumEntries;
    }
    if( upper != -1 ) {
      Indices[NumEntries] = upper;
      Values[NumEntries] = off_upper;
      ++NumEntries;
    }
    // put the off-diagonal entries
    A->InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
    // Put in the diagonal entry
    A->InsertGlobalValues(MyGlobalElements[i], 1, &diag, MyGlobalElements+i);
  }

  // put matrix in local ordering
  A->FillComplete();

  delete [] Indices;
  delete [] Values;
  delete    Map;

  return A;

} /* createJacobian */

// ======================================================================
// This class contians the main definition of the nonlinear problem at
// hand. A method is provided to compute F(x) for a given x, and another
// method to update the entries of the Jacobian matrix, for a given x.
// As the Jacobian matrix J can be written as
//    J = L + diag(lambda*exp(x[i])),
// where L corresponds to the discretization of a Laplacian, and diag
// is a diagonal matrix with lambda*exp(x[i]). Basically, to update
// the jacobian we simply update the diagonal entries. Similarly, to compute
// F(x), we reset J to be equal to L, then we multiply it by the
// (distributed) vector x, then we add the diagonal contribution
// ======================================================================
```

```
class PDEProblem {

public:

  // constructor. Requires the number of nodes along the x-axis
  // and y-axis, the value of lambda, and the Epetra_Communicator
  // (to define a Map, which is a linear map in this case)
  PDEProblem(const int nx, const int ny, const double lambda,
       const Epetra_Comm * Comm) :
    nx_(nx), ny_(ny), lambda_(lambda)
  {
    hx_ = 1.0/(nx_-1);
    hy_ = 1.0/(ny_-1);
    Matrix_ = CreateLaplacian(nx_,ny_,Comm);
  }

  // destructor
  ~PDEProblem()
  {
    delete Matrix_;
  }

  // compute F(x)
  void ComputeF(const Epetra_Vector & x, Epetra_Vector & f)
  {
    // reset diagonal entries
    double diag      =  2.0/(hx_*hx_) + 2.0/(hy_*hy_);

    int NumMyElements = Matrix_->Map().NumMyElements();
    // get update list
    int * MyGlobalElements = Matrix_->Map().MyGlobalElements( );

    for( int i=0 ; i<NumMyElements; ++i ) {
      // Put in the diagonal entry
      Matrix_->ReplaceGlobalValues(MyGlobalElements[i], 1, &diag, MyGlobalElement
      s+i);
    }
    // matrix-vector product (intra-processes communication occurs
    // in this call)
    Matrix_->Multiply(false,x,f);

    // add diagonal contributions
    for( int i=0 ; i<NumMyElements; ++i ) {
      // Put in the diagonal entry
      f[i] += lambda_*exp(x[i]);
    }
  }

  // update the Jacobian matrix for a given x
  void UpdateJacobian(const Epetra_Vector & x)
  {

    double diag      =  2.0/(hx_*hx_) + 2.0/(hy_*hy_);

    int NumMyElements = Matrix_->Map().NumMyElements();
    // get update list
    int * MyGlobalElements = Matrix_->Map().MyGlobalElements( );

    for( int i=0 ; i<NumMyElements; ++i ) {
      // Put in the diagonal entry
      double newdiag = diag + lambda_*exp(x[i]);
```

```
      Matrix_->ReplaceGlobalValues(MyGlobalElements[i], 1,
           &newdiag, MyGlobalElements+i);
    }

  }

  // returns a pointer to the internally stored matrix
  Epetra_CrsMatrix * GetMatrix()
  {
    return Matrix_;
  }

private:

  int nx_, ny_;
  double hx_, hy_;
  Epetra_CrsMatrix * Matrix_;
  double lambda_;

}; /* class PDEProblem */

// ======================================================================
// This is the main NOX class for this example. Here we define
// the interface between the nonlinear problem at hand, and NOX.
// The constructor accepts a PDEProblem object. Using a pointer
// to this object, we can update the Jacobian and compute F(x),
// using the definition of our problem. This interface is bit
// crude: For instance, no PrecMatrix nor Preconditioner is specified.
// ======================================================================

class SimpleProblemInterface : public NOX::Epetra::Interface::Required,
                               public NOX::Epetra::Interface::Jacobian
{

public:

  SimpleProblemInterface( PDEProblem * Problem ) :
    Problem_(Problem) {};

  ~SimpleProblemInterface()
  {
  };

  bool computeF(const Epetra_Vector & x, Epetra_Vector & f,
                NOX::Epetra::Interface::Required::FillType F )
  {
    Problem_->ComputeF(x,f);
    return true;
  };

  bool computeJacobian(const Epetra_Vector & x, Epetra_Operator & Jac)
  {

    Problem_->UpdateJacobian(x);
    return true;
  }

  bool computePrecMatrix(const Epetra_Vector & x, Epetra_RowMatrix & M)
  {
    cout << "*ERR* SimpleProblem::preconditionVector()\n";
    cout << "*ERR* don't use explicit preconditioning" << endl;
    throw 1;
```

```
  }

  bool computePreconditioner(const Epetra_Vector & x, Epetra_Operator & O)
  {
    cout << "*ERR* SimpleProblem::preconditionVector()\n";
    cout << "*ERR* don't use explicit preconditioning" << endl;
    throw 1;
  }

private:

  PDEProblem * Problem_;

}; /* class SimpleProblemInterface */

// =========== //
// main driver //
// =========== //

int main( int argc, char **argv )
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // define the parameters of the nonlinear PDE problem
  int nx = 5;
  int ny = 6;
  double lambda = 1.0;

  PDEProblem Problem(nx,ny,lambda,&Comm);

  // starting solution, here a zero vector
  Epetra_Vector InitialGuess(Problem.GetMatrix()->Map());
  InitialGuess.PutScalar(0.0);

  // Set up the problem interface
  Teuchos::RCP<SimpleProblemInterface> interface =
    Teuchos::rcp(new SimpleProblemInterface(&Problem) );

  // Create the top level parameter list
  Teuchos::RCP<Teuchos::ParameterList> nlParamsPtr =
    Teuchos::rcp(new Teuchos::ParameterList);
  Teuchos::ParameterList& nlParams = *(nlParamsPtr.get());

  // Set the nonlinear solver method
  nlParams.set("Nonlinear Solver", "Line Search Based");

  // Set the printing parameters in the "Printing" sublist
  Teuchos::ParameterList& printParams = nlParams.sublist("Printing");
  printParams.set("MyPID", Comm.MyPID());
  printParams.set("Output Precision", 3);
  printParams.set("Output Processor", 0);
  printParams.set("Output Information",
      NOX::Utils::OuterIteration +
      NOX::Utils::OuterIterationStatusTest +
      NOX::Utils::InnerIteration +
      NOX::Utils::Parameters +
```

```
    NOX::Utils::Details +
    NOX::Utils::Warning);

// start definition of nonlinear solver parameters
// Sublist for line search
Teuchos::ParameterList& searchParams = nlParams.sublist("Line Search");
searchParams.set("Method","More'-Thuente");

// Sublist for direction
Teuchos::ParameterList& dirParams = nlParams.sublist("Direction");
dirParams.set("Method", "Newton");

Teuchos::ParameterList& newtonParams = dirParams.sublist("Newton");
newtonParams.set("Forcing Term Method", "Constant");

// Sublist for linear solver for the Newton method
Teuchos::ParameterList& lsParams = newtonParams.sublist("Linear Solver");
lsParams.set("Aztec Solver", "GMRES");
lsParams.set("Max Iterations", 800);
lsParams.set("Tolerance", 1e-4);
lsParams.set("Output Frequency", 50);
lsParams.set("Aztec Preconditioner", "ilu");

Teuchos::RCP<Epetra_CrsMatrix> A = Teuchos::rcp( Problem.GetMatrix(), false );

Teuchos::RCP<NOX::Epetra::Interface::Required> iReq = interface;
Teuchos::RCP<NOX::Epetra::Interface::Jacobian> iJac = interface;
Teuchos::RCP<NOX::Epetra::LinearSystemAztecOO> linSys =
  Teuchos::rcp(new NOX::Epetra::LinearSystemAztecOO(printParams, lsParams,
               iReq,
               iJac, A,
               InitialGuess));

// Need a NOX::Epetra::Vector for constructor
NOX::Epetra::Vector noxInitGuess(InitialGuess, NOX::DeepCopy);
Teuchos::RCP<NOX::Epetra::Group> grpPtr =
  Teuchos::rcp(new NOX::Epetra::Group(printParams,
        iReq,
        noxInitGuess,
        linSys));

// Set up the status tests
Teuchos::RCP<NOX::StatusTest::NormF> testNormF =
  Teuchos::rcp(new NOX::StatusTest::NormF(1.0e-4));
Teuchos::RCP<NOX::StatusTest::MaxIters> testMaxIters =
  Teuchos::rcp(new NOX::StatusTest::MaxIters(20));
// this will be the convergence test to be used
Teuchos::RCP<NOX::StatusTest::Combo> combo =
  Teuchos::rcp(new NOX::StatusTest::Combo(NOX::StatusTest::Combo::OR,
           testNormF, testMaxIters));

// Create the solver
Teuchos::RCP<NOX::Solver::Generic> solver =
  NOX::Solver::buildSolver(grpPtr, combo, nlParamsPtr);

// Solve the nonlinesar system
NOX::StatusTest::StatusType status = solver->solve();

if( NOX::StatusTest::Converged  != status )
  cout << "\n" << "-- NOX solver converged --" << "\n";
else
  cout << "\n" << "-- NOX solver did not converge --" << "\n";
```

```
  // Print the answer
  cout << "\n" << "-- Parameter List From Solver --" << "\n";
  solver->getList().print(cout);

  // Get the Epetra_Vector with the final solution from the solver
  const NOX::Epetra::Group & finalGroup =
    dynamic_cast<const NOX::Epetra::Group&>(solver->getSolutionGroup());
  const Epetra_Vector & finalSolution =
      (dynamic_cast<const NOX::Epetra::Vector&>(finalGroup.getX())).getEpetraVect
      or();

  if( Comm.MyPID() == 0 ) cout << "Computed solution : " << endl;
  cout << finalSolution;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return(EXIT_SUCCESS);
} /* main */

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif
  puts("Please configure Didasko with:");
  puts("--enable-epetra");
  puts("--enable-ifpack");
  puts("--enable-aztecoo");
  puts("--enable-nox-epetra");
  puts("--enable-nox\n");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return(EXIT_SUCCESS);
}

#endif
```

Chapter 14

ml_ex1

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Two-level domain decomposition preconditioner with AztecO and ML

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_TEUCHOS) && defined(HAVE
      _DIDASKO_TRIUTILS)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_IntVector.h"
#include "Epetra_SerialDenseVector.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_Time.h"
#include "AztecOO.h"
// includes required by ML
#include "ml_include.h"
#include "Epetra_LinearProblem.h"
#include "ml_MultiLevelOperator.h"
#include "ml_epetra_utils.h"

#include "Trilinos_Util_CommandLineParser.h"
#include "Trilinos_Util_CrsMatrixGallery.h"

using namespace Trilinos_Util;

// =========== //
// MAIN DRIVER //
```

```
// ========== //

int main(int argc, char *argv[])
{

#ifdef EPETRA_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // initialize the command line parser
  CommandLineParser CLP(argc,argv);

  // initialize an Gallery object
  CrsMatrixGallery Gallery("", Comm);

  // add default values
  if( CLP.Has("-problem_type") == false ) CLP.Add("-problem_type", "laplace_2d" )
      ;
  if( CLP.Has("-problem_size") == false ) CLP.Add("-problem_size", "100" );

  // initialize the gallery as specified in the command line
  Gallery.Set(CLP);

  // retrive pointers to matrix and linear problem
  Epetra_CrsMatrix * Matrix = Gallery.GetMatrix();
  const Epetra_Map * Map = Gallery.GetMap();

  Epetra_LinearProblem * Problem = Gallery.GetLinearProblem();

  // Construct a solver object for this problem
  AztecOO solver(*Problem);

  // solve with CG (change is matrix is not symmetric)
  solver.SetAztecOption(AZ_solver, AZ_cg);

  // ============================== M L  S E C T I O N ======================
      ===== //

  // Create and set an ML multilevel preconditioner
  ML *ml_handle;

  // Maximum number of levels
  int N_levels = 10;

  // output level
  ML_Set_PrintLevel(3);

  ML_Create(&ml_handle,N_levels);

  // wrap Epetra Matrix into ML matrix (data is NOT copied)
  EpetraMatrix2MLMatrix(ml_handle, 0, Matrix);

  // as we are interested in smoothed aggregation, create a ML_Aggregate object
  // to store the aggregates
  ML_Aggregate *agg_object;
  ML_Aggregate_Create(&agg_object);

  // specify max coarse size (ML will not coarse further is the matrix at a given
       level is
```

```
  // smaller than specified here)
  ML_Aggregate_Set_MaxCoarseSize(agg_object,1);

  // generate the hierady
  N_levels = ML_Gen_MGHierarchy_UsingAggregation(ml_handle, 0,
                                                 ML_INCREASING, agg_object);

  // Set a symmetric Gauss-Seidel smoother for the MG method (change
  // if the matrix is not symmetric)
  ML_Gen_Smoother_SymGaussSeidel(ml_handle, ML_ALL_LEVELS,
                                 ML_BOTH, 1, ML_DEFAULT);

  // generate solver
  ML_Gen_Solver    (ml_handle, ML_MGV, 0, N_levels-1);

  // wrap ML_Operator into Epetra_Operator
  ML_Epetra::MultiLevelOperator  MLop(ml_handle,Comm,*Map,*Map);

  // ====================== E N D    O F  M L  S E C T I O N  ===================
      ====== //

  // set this operator as preconditioner for AztecOO
  solver.SetPrecOperator(&MLop);

  // solve
  solver.Iterate(1550, 1e-12);

  // verify that residual is really small
  double residual, diff;

  Gallery.ComputeResidual(&residual);
  Gallery.ComputeDiffBetweenStartingAndExactSolutions(&diff);

  if( Comm.MyPID() == 0 ) {
    cout << "||b-Ax||_2 = " << residual << endl;
    cout << "||x_exact - x||_2 = " << diff << endl;
  }

  if (residual > 1e-5)
    exit(EXIT_FAILURE);

#ifdef EPETRA_MPI
  MPI_Finalize() ;
#endif
  exit(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
```

```
       "--enable-epetra\n"
       "--enable-ml\n"
       "--enable-triutils");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return 0;
}
#endif
```

Chapter 15

ml_ex2

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Use of ML as a black-box smoothed aggregation preconditioner

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_ML) && defined(HAVE_DIDA
      SKO_TRIUTILS) && defined(HAVE_DIDASKO_AZTECOO) && defined(HAVE_DIDASKO_TEUCHOS)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_RowMatrix.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_LinearProblem.h"
#include "Epetra_Time.h"
#include "AztecOO.h"

// includes required by ML
#include "ml_epetra_preconditioner.h"

#include "Trilinos_Util_CrsMatrixGallery.h"

using namespace Teuchos;
using namespace Trilinos_Util;

#include <iostream>

int main(int argc, char *argv[])
{
```

```
#ifdef EPETRA_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  Epetra_Time Time(Comm);

  // initialize an Gallery object
  CrsMatrixGallery Gallery("laplace_3d", Comm);
  Gallery.Set("problem_size", 1000);

  // retrive pointers to matrix and linear problem
  Epetra_RowMatrix * A = Gallery.GetMatrix();
  Epetra_LinearProblem * Problem = Gallery.GetLinearProblem();

  // Construct a solver object for this problem
  AztecOO solver(*Problem);

  // create the preconditioner object and compute hierarchy
  ML_Epetra::MultiLevelPreconditioner * MLPrec =
    new ML_Epetra::MultiLevelPreconditioner(*A, true);

  // tell AztecOO to use this preconditioner, then solve
  solver.SetPrecOperator(MLPrec);

  solver.SetAztecOption(AZ_solver, AZ_gmres_condnum);
  solver.SetAztecOption(AZ_output, 32);
  solver.SetAztecOption(AZ_kspace, 160);

  int Niters = 500;
  solver.Iterate(Niters, 1e-12);

  // print out some information about the preconditioner
  if( Comm.MyPID() == 0 ) cout << MLPrec->GetOutputList();

  delete MLPrec;

  // compute the real residual

  double residual, diff;
  Gallery.ComputeResidual(&residual);
  Gallery.ComputeDiffBetweenStartingAndExactSolutions(&diff);

  if( Comm.MyPID()==0 ) {
    cout << "||b-Ax||_2 = " << residual << endl;
    cout << "||x_exact - x||_2 = " << diff << endl;
    cout << "Total Time = " << Time.ElapsedTime() << endl;
  }

  if (residual > 1e-5)
    exit(EXIT_FAILURE);
#ifdef EPETRA_MPI
  MPI_Finalize() ;
#endif
  return(EXIT_SUCCESS);
}
#else

#include <stdlib.h>
```

```
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-teuchos\n"
       "--enable-triutils\n"
       "--enable-aztecoo\n"
       "--enable-ml");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);
}
#endif
```

Chapter 16

triutils_ex1

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Read a matrix in HB format

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_TRIUTILS)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Trilinos_Util.h"
#include "Epetra_Comm.h"
#include "Epetra_Map.h"
#include "Epetra_Time.h"
#include "Epetra_BlockMap.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Vector.h"
#include "Epetra_Export.h"
#include "Epetra_CrsMatrix.h"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm (MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  int MyPID = Comm.MyPID();
```

```
  // matrix downloaded from MatrixMarket
  char FileName[] = "../HBMatrices/fidap005.rua";

  Epetra_Map * readMap; // Pointers because of Trilinos_Util_ReadHb2Epetra
  Epetra_CrsMatrix * readA;
  Epetra_Vector * readx;
  Epetra_Vector * readb;
  Epetra_Vector * readxexact;

  // Call routine to read in HB problem
  Trilinos_Util_ReadHb2Epetra(FileName, Comm, readMap, readA, readx,
            readb, readxexact);

  int NumGlobalElements = readMap->NumGlobalElements();

  // Create uniform distributed map
  Epetra_Map map(NumGlobalElements, 0, Comm);

  // Create Exporter to distribute read-in matrix and vectors

  Epetra_Export exporter(*readMap, map);
  Epetra_CrsMatrix A(Copy, map, 0);
  Epetra_Vector x(map);
  Epetra_Vector b(map);
  Epetra_Vector xexact(map);

  Epetra_Time FillTimer(Comm);
  x.Export(*readx, exporter, Add);
  b.Export(*readb, exporter, Add);
  xexact.Export(*readxexact, exporter, Add);
  Comm.Barrier();
  double vectorRedistributeTime = FillTimer.ElapsedTime();
  A.Export(*readA, exporter, Add);
  Comm.Barrier();
  double matrixRedistributeTime = FillTimer.ElapsedTime() - vectorRedistributeTim
      e;
  A.FillComplete();
  Comm.Barrier();
  double fillCompleteTime = FillTimer.ElapsedTime() - matrixRedistributeTime;

  if( MyPID==0 ) {
    cout << "Vector redistribute  time (sec) = "
   << vectorRedistributeTime<< endl;
    cout << "Matrix redistribute time (sec) = "
   << matrixRedistributeTime << endl;
    cout << "Transform to Local  time (sec) = "
   << fillCompleteTime << endl<< endl;
  }

  delete readA;
  delete readx;
  delete readb;
  delete readxexact;
  delete readMap;

#ifdef HAVE_MPI
  MPI_Finalize() ;
#endif

  return(EXIT_SUCCESS);
}
```

```
#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-triutils");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 17

triutils_ex2

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Basic usage of CommandLineParser

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_TRIUTILS)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Trilinos_Util_CommandLineParser.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  // define an Epetra communicator
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  Trilinos_Util::CommandLineParser CLP(argc,argv);

  int nx = CLP.Get("-nx", 123);
  int ny = CLP.Get("-ny", 145);
  double tol = CLP.Get("-tol", 1e-12);
  string solver = CLP.Get("-solver","KLU");
  cout << "nx = " << nx << endl;
  cout << "ny = " << ny << " (default value)" << endl;
```

```
  cout << "tol = " << tol << endl;
  cout << "solver = " << solver << endl;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-triutils");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 18

epetra_ex1

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Basic definition of communicator.
// This code should be run with at least two processes
// However, note that the SAME code works even if Epetra
// has been configured without MPI!

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  // define an Epetra communicator
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // get the proc ID of this process
  int MyPID = Comm.MyPID();

  // get the total number of processes
  int NumProc = Comm.NumProc();
```

```
// output some information to std output
cout << Comm << endl;

// ======================== //
// now some basic MPI calls //
// ------------------------ //

int    ivalue;
double dvalue, dvalue2;
double* dvalues;  dvalues  = new double[NumProc];
double* dvalues2; dvalues2 = new double[NumProc];
int root = 0;

// equivalent to MPI_Barrier

Comm.Barrier();

if (MyPID == root) dvalue = 12.0;

// On input, the root processor contains the list of values
// (in this case, a single value). On exit, all processes will
// have he same list of values. Note that all values must be allocated
// vefore the broadcast

// equivalent to  MPI_Broadcast

Comm.Broadcast(&dvalue, 1, root);

// as before, but with integer values. As C++ can bind to the appropriate
// interface based on argument typing, the type of data is not required.

Comm.Broadcast(&ivalue, 1, root);

// equivalent MPI_Allgather

Comm.GatherAll(dvalues, dvalues2, 1);

// equivalent to MPI_Allreduce with MPI_SUM

dvalue = 1.0*MyPID;

Comm.SumAll( &dvalue, dvalues, 1);

// equivalent to MPI_Allreduce with MPI_SUM

Comm.MaxAll( &dvalue, dvalues, 1);

// equiavant to MPI_Scan with MPI_SUM

dvalue = 1.0 * MyPID;

Comm.ScanSum(&dvalue, &dvalue2, 1);

cout << "On proc " << MyPID << " dvalue2  = " << dvalue2 << endl;

delete[] dvalues;
delete[] dvalues2;

// ======================= //
// Finalize MPI and return //
// ----------------------- //
```

```
#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 19

epetra_ex2

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Definition of Epetra_Map objects

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // global number of points in the map
  int NumGlobalPoint = 4;

  // create a map given the global number of points (any positive number)
  Epetra_Map Map1(NumGlobalPoint,0,Comm);

  // Epetra_Map overloads the << operator
  cout << Map1;
```

```
  // now create a map given the local number of points
  int NumMyPoints = Comm.MyPID();
  Epetra_Map Map2(-1,NumMyPoints,0,Comm);

  cout << Map2;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 20

epetra_ex3

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// define a vector of global dimension 5, split among 2 processes p0 and p1
// int the following way:
//  0 -- 1 -- 2 -- 3 -- 4
//  <-p0->    <----p1--->
// This code must be run with two processes

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include <mpi.h>
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  if (Comm.NumProc() != 2) {
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    return(0);
  }
```

```
  int NumGlobalElements=5;   // global dimension of the problem
  int MyElements = 0;        // local dimension of the problem
  int *MyGlobalElements = 0; // local-to-global map
  int MyPID = Comm.MyPID();  // ID of this process

  if( Comm.NumProc() != 2 ) {
    cerr << "This code must be run with 2 processes\n";
    exit( EXIT_FAILURE );
  }

  // hardwired number of local elements and local-to-global map
  switch( MyPID ) {
  case 0:
    MyElements = 2;
    MyGlobalElements = new int[MyElements];
    MyGlobalElements[0] = 0;
    MyGlobalElements[1] = 1;
    break;
  case 1:
    MyElements = 3;
    MyGlobalElements = new int[MyElements];
    MyGlobalElements[0] = 2;
    MyGlobalElements[1] = 3;
    MyGlobalElements[2] = 4;
    break;
  }

  Epetra_Map Map(NumGlobalElements,MyElements,MyGlobalElements,0,Comm);

  cout << Map;

  delete MyGlobalElements;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(0);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}

#endif
```

Chapter 21

epetra_ex4

```
// Define serial dense vectors
// This code should be run with at least two processes

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_SerialDenseVector.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Total number of elements in vectors
  int NumRows = 5;

  Epetra_SerialDenseVector x;

  // shape this object
  x.Size( NumRows );
```

```
  // set the elements of the vector
  for( int i=0 ; i<NumRows ; ++i )  x[i] = 1.0*i;
  // NOTE: x[i] performs the same operations of x(i);
  // however, the former checks for the bounds, while the latter
  // do such only if Epetra is compiled with -DEPETRA_ARRAY_BOUNDS_CHECK


  // now try to write out of memory
  //int Length = x.Length();

  //x[Length] = 12;

  cout << x;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(0);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}

#endif
```

Chapter 22

epetra_ex5

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Basic operations on vectors. Work on the element vectors.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_IntSerialDenseVector.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Total number of elements in the vector
  int NumElements = 5;

  // Construct a Map with NumElements and index base of 0
  Epetra_Map Map(NumElements, 0, Comm);

  // Create x random vector
```

```
  Epetra_Vector x(Map);

  // get the local size of the vector
  int MyLength = x.MyLength();

  /* First way to define the vector:   */
  /* use the [] operator on the object */

  for( int i=0 ; i<MyLength ; ++i ) x[i] = 1.0*i;

  /* second way: get a copy of the values, modify them, then copy */
  /* back the result into the vector                              */

  // get vector value, put them in 'x_values'
  double *x_values;
  x_values = new double[MyLength];

  x.ExtractCopy( x_values );
  for( int i=0 ; i<MyLength ; ++i )
    cout << "extracted value[" << i << "] = " << x_values[i] << endl;

  int* Indices; Indices = new int[MyLength];

 // now, modify these values
  for( int i=0 ; i<MyLength ; ++i ) {
    x_values[i] *= 10;
    Indices[i] = i;
  }

  // note that the modified values do not affect the internal storage
  // of x.
  cout << x;

  x.ReplaceMyValues(MyLength, 0, x_values, Indices);

  cout << x;

  // free memory and return
  delete[] x_values;
  delete[] Indices;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(0);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
```

```
#endif
```

Chapter 23

epetra_ex6

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
// Questions? Contact Michael A. Heroux (maherou@sandia.gov)
//
// ***********************************************************************
// @HEADER

// Use of ExtractView of Epetra_Vector

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Total number of elements in the vector
  int NumElements = 10;

  // Construct a Map with NumElements and index base of 0
  Epetra_Map Map(NumElements, 0, Comm);

  // Create x as an Epetra_vector
  Epetra_Vector x(Map);
```

```
  // get the local size of the vector
  int MyLength = x.MyLength();

  /* First way to define the vector:   */
  /* use the [] operator on the object */

  for( int i=0 ; i<MyLength ; ++i ) x[i] = 1.0*i;

  // need a double pointer because this works with multi-vectors
  double * pointer;

  x.ExtractView( &pointer );

  for( int i=0 ; i<MyLength ; ++i )
    cout << "on proc " << Comm.MyPID() << ", x["
   << i << "] = " << pointer[i] << endl;

  // now modify the values
  for( int i=0 ; i<MyLength ; ++i )
    pointer[i] *= 10;

  // this affects the object x
  cout << x;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}

#endif
```

Chapter 24

epetra_ex7

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Example of Epetra_MultiVector; use of ExtractView on MultiVector.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_MultiVector.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Total number of elements in the vector
  int NumElements = 10;

  // Construct a Map with NumElements and index base of 0
  Epetra_Map Map(NumElements, 0, Comm);

  // Create x as a 2-component multi-vector
  Epetra_MultiVector x(Map,2);
```

```
  // get the local size of the vector
  int MyLength = x.MyLength();

  /* First way to define the vector:   */
  /* use the [] operator on the object */

  for( int c=0 ; c<x.NumVectors() ; ++c )
    for( int i=0 ; i<MyLength ; ++i ) x[c][i] = 1.0*i+1000*c;

  // need a double pointer because this works with multi-vectors
  double ** pointer;

  x.ExtractView( &pointer );

  for( int c=0 ; c<x.NumVectors() ;++c )
    for( int i=0 ; i<MyLength ; ++i )
      cout << "on proc " << Comm.MyPID() << ", x["
     << i << "] = " << pointer[c][i] << endl;

  // now modify the values
  for( int c=0 ; c<x.NumVectors() ;++c )
    for( int i=0 ; i<MyLength ; ++i )
      pointer[c][i] *= 10;

  cout << x;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(0);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}

#endif
```

Chapter 25

epetra_ex8

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Epetra_Vectors in View mode; use of ResetView.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Total number of elements in the vector
  int NumLocalElements = 10;

  // Construct a Map with NumElements and index base of 0
  Epetra_Map Map(-1, NumLocalElements, 0, Comm);

  // here it is defined a double vector of size NumLocalElements
  // (that is, a size compatible with Map), and it is filled with
```

```
  // some values

  double* values; values = new double[NumLocalElements];
  for( int i=0 ;i<NumLocalElements ; i++ )
    values[i] = 1.0*i;

  // Create x as an Epetra_vector with the View mode, using 'values'
  // as data

  Epetra_Vector x(View, Map, values);

  // now we can change x by modifying values...
  values[0] = 123;

  // this affects the object x
  cout << x;

  // now we can reset the view, and let it point to an other douoble
  // vector (having the same size)

  double* values2; values2 = new double[NumLocalElements];
  for( int i=0 ;i<NumLocalElements ; i++ )
    values2[i] = -1.0*i;

  x.ResetView( values2 );

  cout << x;

  delete[] values;
  delete[] values2;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(0);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}

#endif
```

Chapter 26

epetra_ex9

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Epetra_Export classes
// This code should be run with at least two processes

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_Export.h"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  int NumGlobalElements = 4; // global dimension of the problem

  int NumMyElements; // local nodes
  Epetra_IntSerialDenseVector MyGlobalElements;

  if( Comm.MyPID() == 0 ) {
    NumMyElements = 3;
```

```
    MyGlobalElements.Size(NumMyElements);
    MyGlobalElements[0] = 0;
    MyGlobalElements[1] = 1;
    MyGlobalElements[2] = 2;
  } else {
    NumMyElements = 3;
    MyGlobalElements.Size(NumMyElements);
    MyGlobalElements[0] = 1;
    MyGlobalElements[1] = 2;
    MyGlobalElements[2] = 3;
  }

  // create a map
  Epetra_Map Map(-1,MyGlobalElements.Length(),
     MyGlobalElements.Values(),0, Comm);

  // create a vector based on map
  Epetra_Vector x(Map);
  for( int i=0 ; i<NumMyElements ; ++i )
    x[i] = 10*( Comm.MyPID()+1 );
  cout << x;

  // create a target map, in which all the elements are on proc 0
  int NumMyElements_target;

  if( Comm.MyPID() == 0 )
    NumMyElements_target = NumGlobalElements;
  else
    NumMyElements_target = 0;

  Epetra_Map TargetMap(-1,NumMyElements_target,0,Comm);

  Epetra_Export Exporter(Map,TargetMap);

  // work on vectors
  Epetra_Vector y(TargetMap);

  y.Export(x,Exporter,Add);

  cout << y;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
```

```
#endif
```

Chapter 27

epetra_ex10

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Define dense serial matrices
// This code should be run with one process

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_SerialDenseMatrix.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // declare two dense matrix, whose dimensions are still not specified
  Epetra_SerialDenseMatrix A, B;

  // Total number of rows ans columns for dense matrix A
  int NumRowsA = 2, NumColsA = 2;

  // shape A
  A.Shape( NumRowsA, NumColsA );
```

```
  // set the element of A using the () operator.
  // Note that i is the row-index, and j the column-index
  for( int i=0 ; i<NumRowsA ; ++i )
    for( int j=0 ; j<NumColsA ; ++j )
      A(i,j) = i+100*j;

  // Epetra_SerialDenseMatrix overloads the << operator
  cout << A;

  // get matrix norms
  cout << "Inf norm of A = " << A.OneNorm() << endl;
  cout << "One norm of A = " << A.InfNorm() << endl;

  // now define an other matrix, B, for matrix multiplication
  int NumRowsB = 2, NumColsB=1;
  B.Shape(NumRowsB, NumColsB);

  // enter the values of B
  for( int i=0 ; i<NumRowsB ; ++i )
    for( int j=0 ; j<NumColsB ; ++j )
      B(i,j) = 11.0+i+100*j;

  cout << B;

  // define the matrix which will hold A * B
  Epetra_SerialDenseMatrix AtimesB;

  // same number of rows than A, same columns than B
  AtimesB.Shape(NumRowsA,NumColsB);

  // A * B
  AtimesB.Multiply('N','N',1.0, A, B, 0.0);
  cout << AtimesB;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 28

epetra_ex11

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Basic definition of communicator.
// This code should be run with one process

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include <iostream>
#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_SerialDenseVector.h"
#include "Epetra_SerialDenseMatrix.h"
#include "Epetra_SerialDenseSolver.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Total number of elements in vectors, can be any positive number
  int NumRows = 5;

  Epetra_SerialDenseVector x, b;
  x.Size( NumRows );
```

```
  b.Size( NumRows );

  // set the elements of the vector
  for( int i=0 ; i<NumRows ; ++i ) b[i] = 1.0, x[i]=0.0;

  Epetra_SerialDenseMatrix A, A2;
  A.Shape( NumRows, NumRows );
  A2.Shape( NumRows, NumRows ); // A2 is a copy of A

  // Hilbert matrix (ill-conditioned)
  for( int i=0 ; i<NumRows ; ++i )
    for( int j=0 ; j<NumRows ; ++j )
      A(i,j) = 1.0/(i+j+2);

  cout<< A;

  // set up the solver
  Epetra_SerialDenseSolver Problem;
  Problem.SetMatrix( A );
  Problem.SetVectors( x, b );

  A2 = A;
  // we make a copy of A because Problem.Solve() will
  // overwrite A with its LU decomposition. Try with
  // cout << A after the following invocation

  b.Multiply('N','N',1.0, A2, x, 0.0);

  cout << "A * x = \n" << b;

  double rcond;
  Problem.ReciprocalConditionEstimate(rcond);
  cout << "The (estimated) condition number of A is " << 1/rcond << endl;

  Problem.SetMatrix( A2 );
  Problem.Invert();
  cout << "The inverse of A is\n";
  cout << A2;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 29

epetra_ex12

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// this example creates a tridiagonal matrix of type
//
//     |  2  -1            |
//     | -1   2   -1       |
// A = |      ...  ... ... |
//     |            -1  2  |

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // set global dimension of the matrix to 5, could be any number
  int NumGlobalElements = 5;
```

```
// create a map
Epetra_Map Map(NumGlobalElements,0,Comm);

// local number of rows
int NumMyElements = Map.NumMyElements();

// get update list
int * MyGlobalElements = Map.MyGlobalElements( );

// Create an integer vector NumNz that is used to build the Petra Matrix.
// NumNz[i] is the Number of OFF-DIAGONAL term for the ith global equation
// on this processor

int * NumNz = new int[NumMyElements];

// We are building a tridiagonal matrix where each row has (-1 2 -1)
// So we need 2 off-diagonal terms (except for the first and last equation)

for ( int i=0; i<NumMyElements; i++)
  if (MyGlobalElements[i]==0 || MyGlobalElements[i] == NumGlobalElements-1)
    NumNz[i] = 2;
  else
    NumNz[i] = 3;

// Create a Epetra_Matrix
Epetra_CrsMatrix A(Copy,Map,NumNz);
// (NOTE: constructor 'Epetra_CrsMatrix A(Copy,Map,3);' was ok too.)

// Add  rows one-at-a-time
// Need some vectors to help
// Off diagonal Values will always be -1, diagonal term 2

double *Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int *Indices = new int[2];
double two = 2.0;
int NumEntries;

for( int i=0 ; i<NumMyElements; ++i ) {
  if (MyGlobalElements[i]==0) {
Indices[0] = 1;
NumEntries = 1;
  } else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2;
    NumEntries = 1;
  } else {
    Indices[0] = MyGlobalElements[i]-1;
    Indices[1] = MyGlobalElements[i]+1;
    NumEntries = 2;
  }
  A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
  // Put in the diagonal entry
  A.InsertGlobalValues(MyGlobalElements[i], 1, &two, MyGlobalElements+i);
}

// Finish up, trasforming the matrix entries into local numbering,
// to optimize data transfert during matrix-vector products
A.FillComplete();

// build up two distributed vectors q and z, and compute
// q = A * z
```

```
  Epetra_Vector q(A.RowMap());
  Epetra_Vector z(A.RowMap());

  // Fill z with 1's
  z.PutScalar( 1.0 );

  A.Multiply(false, z, q); // Compute q = A*z

  double dotProduct;
  z.Dot( q, &dotProduct );

  if( Comm.MyPID() == 0 )
    cout << "q dot z = " << dotProduct << endl;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  delete NumNz;

  return( EXIT_SUCCESS );

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 30

epetra_ex13

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// A simple distributed 2D finite element code (Laplacian)
// using the grid
//
//     3      4      5
//     o------o------o
//     | (0) /| (2) /|
//     |    / |    / |
//     |   /  |   /  |
//     |  /   |  /   |
//     | /    | /    |
//     |/ (1) |/ (3) |
//     o------o------o
//     0      1      2
//
// processor 0 hosts the nodes 0, 3,and 4, while processor 1
// hosts 1, 2, and 5.
// Note: The grid information are hardwired, but they can easily
// be replaced by I/O functions.
// This code must be run with two processes.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"
```

```
#include "Epetra_Import.h"
#include "Epetra_SerialDenseMatrix.h"

// function declaration
void compute_loc_matrix( double *x_triangle, double *y_triangle,
       Epetra_SerialDenseMatrix &Ke );
int find( const int list[], const int length, const int index);

// main driver
int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  if (Comm.NumProc() != 2) {
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    return(0);
  }

  int NumMyElements = 0;         // NODES assigned to this processor
  int NumMyExternalElements = 0; // nodes used by this proc, but not hosted
  int NumMyTotalElements = 0;
  int FE_NumMyElements = 0;      // TRIANGLES assigned to this processor
  int * MyGlobalElements = 0;    // nodes assigned to this processor
  Epetra_IntSerialDenseMatrix T; // store the grid connectivity

  int MyPID=Comm.MyPID();

  cout << MyPID << endl;

  switch( MyPID ) {

  case 0:
    NumMyElements = 3;
    NumMyExternalElements = 2;
    NumMyTotalElements = NumMyElements + NumMyExternalElements;
    FE_NumMyElements = 3;

    MyGlobalElements = new int[NumMyTotalElements];
    MyGlobalElements[0] = 0;
    MyGlobalElements[1] = 4;
    MyGlobalElements[2] = 3;
    MyGlobalElements[3] = 1;
    MyGlobalElements[4] = 5;

    break;
  case 1:
    NumMyElements = 3;
    NumMyExternalElements = 2;
    NumMyTotalElements = NumMyElements + NumMyExternalElements;
    FE_NumMyElements = 3;

    MyGlobalElements = new int[NumMyTotalElements];
    MyGlobalElements[0] = 1;
    MyGlobalElements[1] = 2;
```

```
  MyGlobalElements[2] = 5;
  MyGlobalElements[3] = 0;
  MyGlobalElements[4] = 4;
  break;

}

// build Map corresponding to update
Epetra_Map Map(-1,NumMyElements,MyGlobalElements,0,Comm);

// vector containing coordinates BEFORE exchanging external nodes
Epetra_Vector CoordX_noExt(Map);
Epetra_Vector CoordY_noExt(Map);

switch( MyPID ) {

case 0:
  T.Shape(3,FE_NumMyElements);

  // fill x-coordinates
  CoordX_noExt[0] = 0.0;
  CoordX_noExt[1] = 1.0;
  CoordX_noExt[2] = 0.0;
  // fill y-coordinates
  CoordY_noExt[0] = 0.0;
  CoordY_noExt[1] = 1.0;
  CoordY_noExt[2] = 1.0;
  // fill connectivity
  T(0,0) = 0; T(0,1) = 4; T(0,2) = 3;
  T(1,0) = 0; T(1,1) = 1; T(1,2) = 4;
  T(2,0) = 4; T(2,1) = 1; T(2,2) = 5;
  break;

case 1:

  T.Shape(3,FE_NumMyElements);

  // fill x-coordinates
  CoordX_noExt[0] = 1.0;
  CoordX_noExt[1] = 2.0;
  CoordX_noExt[2] = 2.0;
  // fill y-coordinates
  CoordY_noExt[0] = 0.0;
  CoordY_noExt[1] = 0.0;
  CoordY_noExt[2] = 1.0;
  // fill connectivity
  T(0,0) = 0; T(0,1) = 1; T(0,2) = 4;
  T(1,0) = 1; T(1,1) = 5; T(1,2) = 4;
  T(2,0) = 1; T(2,1) = 2; T(2,2) = 5;
  break;
}

// - - - - - - - - - - - - - - - - - - - - //
// E X T E R N A L   N O D E S   S E T U P //
// - - - - - - - - - - - - - - - - - - - - //

// build target map to exchange the valus of external nodes
Epetra_Map TargetMap(-1,NumMyTotalElements,
      MyGlobalElements, 0, Comm);
// !@# rename Map -> SourceMap ?????
Epetra_Import Importer(TargetMap,Map);
Epetra_Vector CoordX(TargetMap);
```

```
Epetra_Vector CoordY(TargetMap);

CoordX.Import(CoordX_noExt,Importer,Insert);
CoordY.Import(CoordY_noExt,Importer,Insert);

// now CoordX_noExt and CoordY_noExt are no longer required
// NOTE: better to construct CoordX and CoordY as MultiVector

// - - - - - - - - - - - - - //
// M A T R I X   S E T U P //
// - - - - - - - - - - - - - //

// build the CRS matrix corresponding to the grid
// some vectors are allocated
const int MaxNnzRow = 5;

Epetra_CrsMatrix A(Copy,Map,MaxNnzRow);

int Element, MyRow, GlobalRow, GlobalCol, i, j, k;
Epetra_IntSerialDenseMatrix Struct; // temp to create the matrix connectivity
Struct.Shape(NumMyElements,MaxNnzRow);
for( i=0 ; i<NumMyElements ; ++i )
  for( j=0 ; j<MaxNnzRow ; ++j )
    Struct(i,j) = -1;

// cycle over all the finite elements
for( Element=0 ; Element<FE_NumMyElements ; ++Element ) {
  // cycle over each row
  for( i=0 ; i<3 ; ++i ) {
    // get the global and local number of this row
    GlobalRow = T(Element,i);
    MyRow = A.LRID(GlobalRow);
    if( MyRow != -1 ) { // only rows stored on this proc
// cycle over the columns
for( j=0 ; j<3 ; ++j ) {
  // get the global number only of this column
  GlobalCol = T(Element,j);
  // look if GlobalCol was already put in Struct
  for( k=0 ; k<MaxNnzRow ; ++k ) {
    if( Struct(MyRow,k) == GlobalCol ||
  Struct(MyRow,k) == -1 ) break;
  }
  if( Struct(MyRow,k) == -1 ) { // new entry
    Struct(MyRow,k) = GlobalCol;
  } else if( Struct(MyRow,k) != GlobalCol ) {
    // maybe not enough space has beenn allocated
    cerr << "ERROR: not enough space for element "
   << GlobalRow << "," << GlobalCol << endl;
    return( 0 );
  }
}
    }
  }
}

int * Indices = new int [MaxNnzRow];
double * Values  = new double [MaxNnzRow];
for( i=0 ; i<MaxNnzRow ; ++i ) Values[i] = 0.0;

// now use Struct to fill build the matrix structure
for( int Row=0 ; Row<NumMyElements ; ++Row ) {
  int Length = 0;
```

```
  for( int j=0 ; j<MaxNnzRow ; ++j ) {
    if( Struct(Row,j) == -1 ) break;
    Indices[Length] = Struct(Row,j);
    Length++;
  }
  GlobalRow = MyGlobalElements[Row];
  A.InsertGlobalValues(GlobalRow, Length, Values, Indices);
}

// replace global numbering with local one in T
for( int Element=0 ; Element<FE_NumMyElements ; ++Element ) {
  for( int i=0 ; i<3 ; ++i ) {
    int global = T(Element,i);
    int local = find(MyGlobalElements,NumMyTotalElements,
    global);
    if( global == -1 ) {
cerr << "ERROR\n";
return( EXIT_FAILURE );
    }
    T(Element,i) = local;
  }
}

// - - - - - - - - - - - - - - - //
// M A T R I X   F I L L - I N //
// - - - - - - - - - - - - - - - //

// room for the local matrix
Epetra_SerialDenseMatrix Ke;
Ke.Shape(3,3);

// now fill the matrix
for(  int Element=0 ; Element<FE_NumMyElements ; ++Element ) {
  // variables used inside
  int GlobalRow;
  int MyRow;
  int GlobalCol;
  double x_triangle[3];
  double y_triangle[3];
  // get the spatial coordinate of each local node
  for( int i=0 ; i<3 ; ++i ) {
    MyRow = T(Element,i);
    y_triangle[i] = CoordX[MyRow];
    x_triangle[i] = CoordY[MyRow];
  }
  // compute the local matrix for Element

  compute_loc_matrix( x_triangle, y_triangle,Ke );

  // insert it in the global one
  // cycle over each row
  for( int i=0 ; i<3 ; ++i ) {
    // get the global and local number of this row
    MyRow = T(Element,i);
    if( MyRow < NumMyElements ) {
for( int j=0 ; j<3 ; ++j ) {
  // get global column number
  GlobalRow = MyGlobalElements[MyRow];
  GlobalCol = MyGlobalElements[T(Element,j)];
  A.SumIntoGlobalValues(GlobalRow,1,&(Ke(i,j)),&GlobalCol);
}
    }
```

```
    }
  }

  A.FillComplete();

  // - - - - - - - - - - - - - - - //
  // R H S  &  S O L U T I O N //
  // - - - - - - - - - - - - - - - //

  Epetra_Vector x(Map), b(Map);
  x.Random(); b.PutScalar(0.0);

  // Solution can be obtained using Aztecoo

  // free memory before leaving
  delete MyGlobalElements;
  delete Indices;
  delete Values;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

} /* main */

int find( const int list[], const int length, const int index)
{

  int pos=-1;
  for( int i=0 ; i<length ; ++i )
    if( list[i] == index ) {
      pos = i;
      break;
    }

  return pos;

} /* find */

void compute_loc_matrix( double *x_triangle, double *y_triangle,
       Epetra_SerialDenseMatrix & Ke )
{

  int    ii, jj;
  double det_J;
  double xa, ya, xb, yb, xc, yc;

  xa = x_triangle[0];
  xb = x_triangle[1];
  xc = x_triangle[2];
  ya = y_triangle[0];
  yb = y_triangle[1];
  yc = y_triangle[2];

  Ke(0,0) = (yc-yb)*(yc-yb) + (xc-xb)*(xc-xb);
  Ke(0,1) = (yc-yb)*(ya-yc) + (xc-xb)*(xa-xc);
  Ke(0,2) = (yb-ya)*(yc-yb) + (xb-xa)*(xc-xb);
  Ke(1,0) = (yc-yb)*(ya-yc) + (xc-xb)*(xa-xc);
  Ke(1,1) = (yc-ya)*(yc-ya) + (xc-xa)*(xc-xa);
  Ke(1,2) = (ya-yc)*(yb-ya) + (xa-xc)*(xb-xa);
```

```
  Ke(2,0) = (yb-ya)*(yc-yb) + (xb-xa)*(xc-xb);
  Ke(2,1) = (ya-yc)*(yb-ya) + (xa-xc)*(xb-xa);
  Ke(2,2) = (yb-ya)*(yb-ya) + (xb-xa)*(xb-xa);

  det_J = (xb-xa)*(yc-ya)-(xc-xa)*(yb-ya);
  det_J = 2*det_J;
  if( det_J<0 ) det_J *=-1;

  for (ii = 0; ii < 3; ii++) {
    for (jj = 0; jj < 3; jj++) {
      Ke(ii,jj) = Ke(ii,jj) / det_J;
    }
  }

  return;

} /* compute_loc_matrix */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 31

epetra_ex14

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Print out some information about a CRS matrix
#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "float.h"
#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"

// ============================================================
// define a class, derived from Epetra_CrsMatrix, which
// initializes the matrix entires. User has to provide
// a valid Epetra_Map in the constructor, plus the diagonal
// value, and the sub- and super-diagonal values.
// ============================================================

class TridiagonalCrsMatrix : public Epetra_CrsMatrix {

public:

  TridiagonalCrsMatrix(const Epetra_Map & Map,
            double a,
            double diag, double c) :
    Epetra_CrsMatrix(Copy,Map,3)
  {
```

```
    // global number of rows
    int NumGlobalElements = Map.NumGlobalElements();
    // local number of rows
    int NumMyElements = Map.NumMyElements();
    // get update list
    int * MyGlobalElements = new int [NumMyElements];
    Map.MyGlobalElements( MyGlobalElements );

    // Add  rows one-at-a-time
    // Need some vectors to help
    // Off diagonal Values will always be -1

    double *Values = new double[2];
    Values[0] = a; Values[1] = c;
    int *Indices = new int[2];
    int NumEntries;

    for( int i=0 ; i<NumMyElements; ++i ) {
      if (MyGlobalElements[i]==0) {
  Indices[0] = 1;
  NumEntries = 1;
      } else if (MyGlobalElements[i] == NumGlobalElements-1) {
  Indices[0] = NumGlobalElements-2;
  NumEntries = 1;
      } else {
  Indices[0] = MyGlobalElements[i]-1;
  Indices[1] = MyGlobalElements[i]+1;
  NumEntries = 2;
      }
      InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
      // Put in the diagonal entry
      InsertGlobalValues(MyGlobalElements[i], 1, &diag, MyGlobalElements+i);
    }

    // Finish up
    FillComplete();

  }

};

// ======================================================
// This function print out some information about the input
// matrix. This includes number of rows and columns, some norms,
// few statistics about the nonzero structure of the matrix.
// Some details about the Trilinos storage of the matrix
// are also reported.
//
// Return code:        true if matrix has been printed out
// ------------        false otherwise
//
// Parameters:
// -----------
//
// - Epetra_CrsMatrix  reference to the ditributed CrsMatrix to
//                     print out
// - os                output stream (can be cout)
//=============================================================

bool CrsMatrixInfo( const Epetra_CrsMatrix & A,
        ostream & os )
```

```
{

  int MyPID = A.Comm().MyPID();

  // take care that matrix is already trasformed
  bool IndicesAreGlobal = A.IndicesAreGlobal();
  if( IndicesAreGlobal == true ) {
    if( MyPID == 0 ) {
      os << "WARNING : matrix must be transformed to local\n";
      os << "          before calling CrsMatrixInfo\n";
      os << "          Now returning...\n";
    }
    return false;
  }

  int NumGlobalRows = A.NumGlobalRows();
  int NumGlobalNonzeros = A.NumGlobalNonzeros();
  int NumGlobalCols = A.NumGlobalCols();
  double NormInf = A.NormInf();
  double NormOne = A.NormOne();
  int NumGlobalDiagonals = A.NumGlobalDiagonals();
  int GlobalMaxNumEntries = A.GlobalMaxNumEntries();
  int IndexBase = A.IndexBase();
  bool StorageOptimized = A.StorageOptimized();
  bool LowerTriangular = A.LowerTriangular();
  bool UpperTriangular = A.UpperTriangular();
  bool NoDiagonal = A.NoDiagonal();

  // these variables identifies quantities I have to compute,
  // since not provided by Epetra_CrsMatrix

  double MyFrobeniusNorm( 0.0 ), FrobeniusNorm( 0.0 );
  double MyMinElement( DBL_MAX ), MinElement( DBL_MAX );
  double MyMaxElement( DBL_MIN ), MaxElement( DBL_MIN );
  double MyMinAbsElement( DBL_MAX ), MinAbsElement( DBL_MAX );
  double MyMaxAbsElement( 0.0 ), MaxAbsElement( 0.0 );

  int NumMyRows = A.NumMyRows();
  int * NzPerRow = new int[NumMyRows];
  int Row; // iterator on rows
  int Col; // iterator on cols
  int MaxNumEntries = A.MaxNumEntries();
  double * Values = new double[MaxNumEntries];
  int * Indices = new int[MaxNumEntries];
  double Element, AbsElement; // generic nonzero element and its abs value
  int NumEntries;
  double * Diagonal = new double [NumMyRows];
  // SumOffDiagonal is the sum of absolute values for off-diagonals
  double * SumOffDiagonal = new double [NumMyRows];
  for( Row=0 ;  Row<NumMyRows ; ++Row ) {
    SumOffDiagonal[Row] = 0.0;
  }
  int * IsDiagonallyDominant = new int [NumMyRows];
  int GlobalRow;

  // cycle over all matrix elements
  for( Row=0 ; Row<NumMyRows ; ++Row ) {
    GlobalRow = A.GRID(Row);
    NzPerRow[Row] = A.NumMyEntries(Row);
    A.ExtractMyRowCopy(Row,NzPerRow[Row],NumEntries,Values,Indices);
    for( Col=0 ; Col<NumEntries ; ++Col ) {
      Element = Values[Col];
```

```
    AbsElement = abs(Element);
    if( Element<MyMinElement ) MyMinElement = Element;
    if( Element>MyMaxElement ) MyMaxElement = Element;
    if( AbsElement<MyMinAbsElement ) MyMinAbsElement = AbsElement;
    if( AbsElement>MyMaxAbsElement ) MyMaxAbsElement = AbsElement;
    if( Indices[Col] == Row ) Diagonal[Row] = Element;
    else
SumOffDiagonal[Row] += abs(Element);
    MyFrobeniusNorm += pow(Element,2);
  }
}

// analise storage per row
int MyMinNzPerRow( NumMyRows ), MinNzPerRow( NumMyRows );
int MyMaxNzPerRow( 0 ), MaxNzPerRow( 0 );

for( Row=0 ; Row<NumMyRows ; ++Row ) {
  if( NzPerRow[Row]<MyMinNzPerRow ) MyMinNzPerRow=NzPerRow[Row];
  if( NzPerRow[Row]>MyMaxNzPerRow ) MyMaxNzPerRow=NzPerRow[Row];
}

// a test to see if matrix is diagonally-dominant

int MyDiagonalDominance( 0 ), DiagonalDominance( 0 );
int MyWeakDiagonalDominance( 0 ), WeakDiagonalDominance( 0 );

for( Row=0 ; Row<NumMyRows ; ++Row ) {
  if( abs(Diagonal[Row])>SumOffDiagonal[Row] )
    ++MyDiagonalDominance;
  else if( abs(Diagonal[Row])==SumOffDiagonal[Row] )
    ++MyWeakDiagonalDominance;
}

// reduction operations

A.Comm().SumAll(&MyFrobeniusNorm, &FrobeniusNorm, 1);
A.Comm().MinAll(&MyMinElement, &MinElement, 1);
A.Comm().MaxAll(&MyMaxElement, &MaxElement, 1);
A.Comm().MinAll(&MyMinAbsElement, &MinAbsElement, 1);
A.Comm().MaxAll(&MyMaxAbsElement, &MaxAbsElement, 1);
A.Comm().MinAll(&MyMinNzPerRow, &MinNzPerRow, 1);
A.Comm().MaxAll(&MyMaxNzPerRow, &MaxNzPerRow, 1);
A.Comm().SumAll(&MyDiagonalDominance, &DiagonalDominance, 1);
A.Comm().SumAll(&MyWeakDiagonalDominance, &WeakDiagonalDominance, 1);

// free memory

delete Values;
delete Indices;
delete Diagonal;
delete SumOffDiagonal;
delete IsDiagonallyDominant;
delete NzPerRow;

// simply no output for MyPID>0, only proc 0 write on os
if( MyPID != 0 ) return true;

os << "*** general Information about the matrix\n";
os << "Number of Global Rows = " << NumGlobalRows << endl;
os << "Number of Global Cols = " << NumGlobalCols << endl;
os << "is the matrix square  = " <<
  ((NumGlobalRows==NumGlobalCols)?"yes":"no") << endl;
```

```
  os << "||A||_\\infty          = " << NormInf << endl;
  os << "||A||_1               = " << NormOne << endl;
  os << "||A||_F               = " << sqrt(FrobeniusNorm) << endl;
  os << "Number of nonzero diagonal entries = "
     << NumGlobalDiagonals
     << "( " << 1.0* NumGlobalDiagonals/NumGlobalRows*100
     << " %)\n";
  os << "Nonzero per row : min = " << MinNzPerRow
     << " average = " << 1.0*NumGlobalNonzeros/NumGlobalRows
     << " max = " << MaxNzPerRow << endl;
  os << "Maximum number of nonzero elements/row = "
     << GlobalMaxNumEntries << endl;
  os << "min( a_{i,j} )      = " << MinElement << endl;
  os << "max( a_{i,j} )      = " << MaxElement << endl;
  os << "min( abs(a_{i,j}) ) = " << MinAbsElement << endl;
  os << "max( abs(a_{i,j}) ) = " << MaxAbsElement << endl;
  os << "Number of diagonal dominant rows        = " << DiagonalDominance
     << " (" << 100.0*DiagonalDominance/NumGlobalRows << " % of total)\n";
  os << "Number of weakly diagonal dominant rows = "
     << WeakDiagonalDominance
     << " (" << 100.0*WeakDiagonalDominance/NumGlobalRows << " % of total)\n";

  os << "*** Information about the Trilinos storage\n";
  os << "Base Index                = " << IndexBase << endl;
  os << "is storage optimized      = "
     << ((StorageOptimized==true)?"yes":"no") << endl;
  os << "are indices global        = "
     << ((IndicesAreGlobal==true)?"yes":"no") << endl;
  os << "is matrix lower triangular = "
     << ((LowerTriangular==true)?"yes":"no") << endl;
  os << "is matrix upper triangular = "
     << ((UpperTriangular==true)?"yes":"no") << endl;
  os << "are there diagonal entries = "
     <<  ((NoDiagonal==false)?"yes":"no") << endl;

  return true;

}

// =========== //
// Main driver //
// =========== //

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // set global dimension to 5, could be any number
  int NumGlobalElements = 5;

  // create a map
  Epetra_Map Map(NumGlobalElements,0,Comm);

  TridiagonalCrsMatrix A( Map, -1.0, 2.0, -1.0);

  // query the matrix
```

```
  CrsMatrixInfo(A, cout);

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 32

epetra_ex15

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Output a CRS matrix in MATLAB format

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"

// ======================================================
// define a class, derived from Epetra_CrsMatrix, which
// initializes the matrix entires. User has to provide
// a valid Epetra_Map in the constructor, plus the diagonal
// value, and the sub- and super-diagonal values.
// ======================================================

class TridiagonalCrsMatrix : public Epetra_CrsMatrix {

public:
  TridiagonalCrsMatrix(const Epetra_Map & Map,
            double a,
            double diag, double c) :
    Epetra_CrsMatrix(Copy,Map,3)
  {

    // global number of rows
```

```
    int NumGlobalElements = Map.NumGlobalElements();
    // local number of rows
    int NumMyElements = Map.NumMyElements();
    // get update list
    int * MyGlobalElements = new int [NumMyElements];
    Map.MyGlobalElements( MyGlobalElements );

    // Add  rows one-at-a-time
    // Need some vectors to help
    // Off diagonal Values will always be -1

    double *Values = new double[2];
    Values[0] = a; Values[1] = c;
    int *Indices = new int[2];
    int NumEntries;

    for( int i=0 ; i<NumMyElements; ++i ) {
      if (MyGlobalElements[i]==0) {
  Indices[0] = 1;
  NumEntries = 1;
      } else if (MyGlobalElements[i] == NumGlobalElements-1) {
  Indices[0] = NumGlobalElements-2;
  NumEntries = 1;
      } else {
  Indices[0] = MyGlobalElements[i]-1;
  Indices[1] = MyGlobalElements[i]+1;
  NumEntries = 2;
      }
      InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
      // Put in the diagonal entry
      InsertGlobalValues(MyGlobalElements[i], 1, &diag, MyGlobalElements+i);
    }

    FillComplete();
  }

};

/* ======== ================ *
 * function CrsMatrix2MATLAB *
 * ======== ================ *
 *
 * Print out a CrsMatrix in a MATLAB format. Each processor prints out
 * its part, starting from proc 0 to proc NumProc-1. The first line of
 * each processor's output states the number of local rows and of
 * local nonzero elements. Output is finished by "End of Matrix Output".
 *
 *
 * Return code:        true if matrix has been printed out
 * ------------        false otherwise
 *
 * Parameters:
 * -----------
 *
 * - Epetra_CrsMatrix  reference to the ditributed CrsMatrix to
 *                     print out
 */

bool CrsMatrix2MATLAB( const Epetra_CrsMatrix & A )

{
```

```
int MyPID = A.Comm().MyPID();
int NumProc = A.Comm().NumProc();

// work only on transformed matrices;
if( A.IndicesAreLocal() == false ) {
  if( MyPID == 0 ) {
    cerr << "ERROR in "<< __FILE__ << ", line " << __LINE__ << endl;
    cerr << "Function CrsMatrix2MATLAB accepts\n";
    cerr << "transformed matrices ONLY. Please call A.FillComplete()\n";
    cerr << "on your matrix A to that purpose.\n";
    cerr << "Now returning...\n";
  }
  return false;
}

int NumMyRows = A.NumMyRows(); // number of rows on this process
int NumNzRow;   // number of nonzero elements for each row
int NumEntries; // number of extracted elements for each row
int NumGlobalRows; // global dimensio of the problem
int GlobalRow;  // row in global ordering
int NumGlobalNonzeros; // global number of nonzero elements

NumGlobalRows = A.NumGlobalRows();
NumGlobalNonzeros = A.NumGlobalNonzeros();

// print out on cout if no filename is provided

int IndexBase = A.IndexBase(); // MATLAB start from 0
if( IndexBase == 0 ) IndexBase = 1;

// write on file the dimension of the matrix

if( MyPID==0 ) {
  cout << "A = spalloc(";
  cout << NumGlobalRows << ',' << NumGlobalRows;
  cout << ',' << NumGlobalNonzeros << ");\n";
}

for( int Proc=0 ; Proc<NumProc ; ++Proc ) {

  if( MyPID == Proc ) {

    cout << "% On proc " << Proc << ": ";
    cout << NumMyRows << " rows and ";
    cout << A.NumMyNonzeros() << " nonzeros\n";

    // cycle over all local rows to find out nonzero elements
    for( int MyRow=0 ; MyRow<NumMyRows ; ++MyRow ) {

GlobalRow = A.GRID(MyRow);

NumNzRow = A.NumMyEntries(MyRow);
double *Values = new double[NumNzRow];
int *Indices = new int[NumNzRow];

A.ExtractMyRowCopy(MyRow, NumNzRow,
       NumEntries, Values, Indices);
// print out the elements with MATLAB syntax
for( int j=0 ; j<NumEntries ; ++j ) {
  cout << "A(" << GlobalRow  + IndexBase
       << "," << A.GCID(Indices[j]) + IndexBase
       << ") = " << Values[j] << ";\n";
```

```
  }

  delete Values;
  delete Indices;
      }

    }
    A.Comm().Barrier();
    if( MyPID == 0 ) {
      cout << " %End of Matrix Output\n";
    }
  }

  return true;

}

// =========== //
// main driver //
// =========== //

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // set global dimension to 5, could be any number
  int NumGlobalElements = 5;

  // define a linear map
  Epetra_Map Map(NumGlobalElements,0,Comm);

  // create the matrix
  TridiagonalCrsMatrix A( Map, -1.0, 2.0, -1.0);

  // output informationto stdout
  CrsMatrix2MATLAB( A );

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
```

```
#endif
```

Chapter 33

epetra_ex16

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Output a vector in MATLAB format

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"

/* ======== ============= *
 * function Vector2MATLAB *
 * ======== ============= *
 *
 * Print out a Epetra_Vector in a MATLAB format. Each processor prints out
 * its part, starting from proc 0 to proc NumProc-1. The first line of
 * each processor's output states the number of local rows and of
 * local nonzero elements. Output is finished by "End of Vector Output".
 *
 * Return code:        true if vector has been printed out
 * -----------         false otherwise
 *
 * Parameters:
 * ----------
 *
 * - Epetra_CrsMatrix  reference to vector
 * - Epetra_Map        reference to map
 */
```

```
bool Vector2MATLAB( const Epetra_Vector & v,
        const Epetra_Map & Map)
{

  int MyPID = Map.Comm().MyPID();
  int NumProc = Map.Comm().NumProc();
  int MyLength = v.MyLength();
  int GlobalLength = v.GlobalLength();

  // print out on cout if no filename is provided

  // write on file the dimension of the matrix

  if( MyPID == 0 ) cout << "v = zeros(" << GlobalLength << ")\n";

  // get update list
  int * MyGlobalElements = Map.MyGlobalElements( );

  int Row;

  for( int Proc=0 ; Proc<NumProc ; ++Proc ) {

    if( MyPID == Proc ) {

      cout << "% On proc " << Proc << ": ";
      cout << MyLength << " rows of ";
      cout << GlobalLength << " elements\n";

      for( Row=0 ; Row<MyLength ; ++Row ) {
  cout << "b(" << MyGlobalElements[Row]
       << ") = " << v[Row] << ";\n";
      }

      if( MyPID == NumProc-1  ) {
  cout << "% End of vector\n";
      }

    }

    Map.Comm().Barrier();
  }

  return true;

} /* Vector2MATLAB */


// =========== //
// main driver //
// =========== //

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif
```

```
  // set global dimension to 5, could be any number
  int NumGlobalElements = 5;

  // create a map
  Epetra_Map Map(NumGlobalElements,0,Comm);
  Epetra_Vector A(Map);

  // fill the vector with random elements
  A.Random();

  // and output it to stdout
  Vector2MATLAB( A, Map );

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 34

epetra_ex17

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Basic definition of communicator.
// This code should be run with at least two processes
// However, note that the SAME code works even if Epetra
// has been configured without MPI!

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  // define an Epetra communicator
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // get the proc ID of this process
  int MyPID = Comm.MyPID();

  // get the total number of processes
  int NumProc = Comm.NumProc();
```

```
// output some information to std output
cout << Comm << endl;

// ======================== //
// now some basic MPI calls //
// ------------------------ //

int    ivalue;
double dvalue, dvalue2;
double* dvalues;  dvalues  = new double[NumProc];
double* dvalues2; dvalues2 = new double[NumProc];
int root = 0;

// equivalent to MPI_Barrier

Comm.Barrier();

if (MyPID == root) dvalue = 12.0;

// On input, the root processor contains the list of values
// (in this case, a single value). On exit, all processes will
// have he same list of values. Note that all values must be allocated
// vefore the broadcast

// equivalent to  MPI_Broadcast

Comm.Broadcast(&dvalue, 1, root);

// as before, but with integer values. As C++ can bind to the appropriate
// interface based on argument typing, the type of data is not required.

Comm.Broadcast(&ivalue, 1, root);

// equivalent MPI_Allgather

Comm.GatherAll(dvalues, dvalues2, 1);

// equivalent to MPI_Allreduce with MPI_SUM

dvalue = 1.0*MyPID;

Comm.SumAll( &dvalue, dvalues, 1);

// equivalent to MPI_Allreduce with MPI_SUM

Comm.MaxAll( &dvalue, dvalues, 1);

// equiavant to MPI_Scan with MPI_SUM

dvalue = 1.0 * MyPID;

Comm.ScanSum(&dvalue, &dvalue2, 1);

cout << "On proc " << MyPID << " dvalue2  = " << dvalue2 << endl;

delete[] dvalues;
delete[] dvalues2;

// ======================= //
// Finalize MPI and return //
// ----------------------- //
```

```
#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 35

epetra_ex18

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Basic operations on vectors. The code defines two vectors and
// compute this dot product.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Total number of elements in the vector
  int NumElements = 5;

  // Construct a Map with NumElements and index base of 0
  Epetra_Map Map(NumElements, 0, Comm);

  // Create x and b vectors
```

```
  Epetra_Vector x(Map);
  Epetra_Vector b(Map);

  // set all the elements to a scalar value
  x.Random( );

  // random numbers
  b.Random();

  // minimum, maximum and average of vector
  double minval, maxval, aveval;
  x.MinValue( &minval );
  x.MaxValue( &maxval );
  x.MeanValue( &aveval );
  cout << " min(x) = " << minval << endl;
  cout << " max(x) = " << maxval << endl;
  cout << " ave(x) = " << aveval << endl;

  // dot product
  double xdotb;
  x.Dot( b, &xdotb );
  cout << "x dot b = " << xdotb << endl;

  // total number of vectors, local and global dimension
  int NVecs, MySize, GloSize;
  NVecs = x.NumVectors();
  MySize = x.MyLength();
  GloSize = x.GlobalLength();
  cout << "Number of vectors = " << NVecs << endl;
  cout << "Local Size = " << MySize << endl;
  cout << "Global Size = " << GloSize << endl;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 36

epetra_ex19

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Create a Crs matrix corresponding to a 2D Laplacian problem
// on a cartesian mesh.

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"

// function declaration

void  get_neighbours( const int i, const int nx, const int ny,
          int & left, int & right,
          int & lower, int & upper);

// =========== //
// main driver //
// =========== //

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
```

```
#else
  Epetra_SerialComm Comm;
#endif

  // number of nodes in the x- and y-direction
  int nx = 5;
  int ny = 6;
  int NumGlobalElements = nx * ny;

  // create a linear map
  Epetra_Map Map(NumGlobalElements,0,Comm);

  // local number of rows
  int NumMyElements = Map.NumMyElements();
  // get update list
  int * MyGlobalElements = Map.MyGlobalElements( );

  // Create an integer vector NumNz that is used to build the Petra Matrix.
  // NumNz[i] is the Number of OFF-DIAGONAL term for the ith global equation
  // on this processor.
  // NOTE: NumNz can be specified to be an interfer, of value 5.
  // However, the procedure here reported is more general, and it is
  // representative of more complex situations, where the number of
  // nonzero per row can vary consistently.

  int * NumNz = new int[NumMyElements];

  double off_left  = -1.0;
  double off_right = -1.0;
  double off_lower = -1.0;
  double off_upper = -1.0;
  double diag      =  4.0;
  int left, right, lower, upper;

  for ( int i=0; i<NumMyElements; i++) {
    NumNz[i] = 1;
    get_neighbours( MyGlobalElements[i], nx, ny,
        left, right, lower, upper);
    if( left  != -1 ) ++NumNz[i];
    if( right != -1 ) ++NumNz[i];
    if( lower != -1 ) ++NumNz[i];
    if( upper != -1 ) ++NumNz[i];
  }

  // Create a Epetra_Matrix
  // create a CRS matrix

  Epetra_CrsMatrix A(Copy,Map,NumNz);

  // Add  rows one-at-a-time

  double Values[4];
  int Indices[4];
  int NumEntries;

  for( int i=0 ; i<NumMyElements; ++i ) {
    int NumEntries=0;
    get_neighbours(  MyGlobalElements[i], nx, ny,
         left, right, lower, upper);
    if( left != -1 ) {
  Indices[NumEntries] = left;
  Values[NumEntries] = off_left;
```

```
  ++NumEntries;
    }
    if( right != -1 ) {
      Indices[NumEntries] = right;
      Values[NumEntries] = off_right;
      ++NumEntries;
    }
    if( lower != -1 ) {
      Indices[NumEntries] = lower;
      Values[NumEntries] = off_lower;
      ++NumEntries;
    }
    if( upper != -1 ) {
      Indices[NumEntries] = upper;
      Values[NumEntries] = off_upper;
      ++NumEntries;
    }
    // put the off-diagonal entries
    A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
    // Put in the diagonal entry
    A.InsertGlobalValues(MyGlobalElements[i], 1, &diag, MyGlobalElements+i);
  }
  cout <<  A;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  delete NumNz;

  return(EXIT_SUCCESS);

}

void  get_neighbours( const int i, const int nx, const int ny,
          int & left, int & right,
          int & lower, int & upper)
{

  int ix, iy;
  ix = i%nx;
  iy = (i - ix)/nx;

  if( ix == 0 )
    left = -1;
  else
    left = i-1;
  if( ix == nx-1 )
    right = -1;
  else
    right = i+1;
  if( iy == 0 )
    lower = -1;
  else
    lower = i-nx;
  if( iy == ny-1 )
    upper = -1;
  else
    upper = i+nx;

  return;
```

```
}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 37

epetra_ex20

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Example of use of Epetra_Time and Epetra_Flops

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_Time.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  bool verbose = (Comm.MyPID() == 0);

  // set global dimension to 5, could be any number
  int NumGlobalElements = 5;
  // create a map
  Epetra_Map Map(NumGlobalElements,0,Comm);
```

```
// local number of rows
int NumMyElements = Map.NumMyElements();
// get update list
int * MyGlobalElements = Map.MyGlobalElements( );

// ============= CONSTRUCTION OF THE MATRIX ============================
// Create a Epetra_Matrix

Epetra_CrsMatrix A(Copy,Map,3);

// Add  rows one-at-a-time

double *Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int *Indices = new int[2];
double two = 2.0;
int NumEntries;

for( int i=0 ; i<NumMyElements; ++i ) {
  if (MyGlobalElements[i]==0) {
Indices[0] = 1;
NumEntries = 1;
  } else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2;
    NumEntries = 1;
  } else {
    Indices[0] = MyGlobalElements[i]-1;
    Indices[1] = MyGlobalElements[i]+1;
    NumEntries = 2;
  }
  A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
  // Put in the diagonal entry
  A.InsertGlobalValues(MyGlobalElements[i], 1, &two, MyGlobalElements+i);
}

// Finish up
A.FillComplete();

// ================ CONSTRUCTION OF VECTORS ======================

// build up two distributed vectors q and z, and compute
// q = A * z
Epetra_Vector q(A.RowMap());
Epetra_Vector z(A.RowMap());

// Fill z with 1's
z.PutScalar( 1.0 );

// ================ USE OF TIME AND FLOPS =========================

Epetra_Flops counter;
A.SetFlopCounter(counter);
Epetra_Time timer(Comm);

A.Multiply(false, z, q); // Compute q = A*z

double elapsed_time = timer.ElapsedTime();
double total_flops =counter.Flops();
if (verbose)
  cout << "Total ops: " << total_flops << endl;
double MFLOPs = total_flops/elapsed_time/1000000.0;
if (verbose)
```

```
    cout << "Total MFLOPs  for mat-vec = " << MFLOPs << endl<< endl;

  double dotProduct;
  z.SetFlopCounter(counter);
  timer.ResetStartTime();
  z.Dot(q, &dotProduct);

  total_flops =counter.Flops();
  if (verbose)
    cout << "Total ops: " << total_flops << endl;

  elapsed_time = timer.ElapsedTime();
  if (elapsed_time != 0.0)
    MFLOPs = (total_flops / elapsed_time) / 1000000.0;
  else
    MFLOPs = 0;

  if (verbose)
  {
    cout << "Total MFLOPs for vec-vec = " << MFLOPs << endl<< endl;
    cout << "q dot z = " << dotProduct << endl;
  }

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( 0 );

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 38

epetra_ex21

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Use of Epetra_Operator.
// This code must be run with one process

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Operator.h"

// ==================== //
// TriDiagonal Operator //
// -------------------- //

class TriDiagonalOperator : public Epetra_Operator
{

public:

  // constructor
  TriDiagonalOperator( double diag_minus_one,
           double diag,
           double diag_plus_one,
           Epetra_Map & Map) :
    Map_(Map),
```

```
  diag_minus_one_(diag_minus_one),
  diag_(diag),
  diag_plus_one_(diag_plus_one)
{}

// application of the tridiagonal operator
int Apply( const Epetra_MultiVector & X,
     Epetra_MultiVector & Y ) const
{
  int Length = X.MyLength();

  // maybe some error checks on MultiVector Lenghts
  // for the future...

  for( int vec=0 ; vec<X.NumVectors() ; ++vec ) {

    // one-dimensional problems here
    if( Length == 1 ) {
Y[vec][0] = diag_ * X[vec][0];
break;
    }

    // more general case (Lenght >= 2)

    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];

    // intermediate rows
    for( int i=1 ; i<Length-1 ; ++i ) {
Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1]
  + diag_minus_one_ * X[vec][i-1];
    }
    // final row
    Y[vec][Length-1] = diag_ * X[vec][Length-1]
+ diag_minus_one_ * X[vec][Length-2];
  }

  return true;
}

// other function
int SetUseTranspose( bool UseTranspose)
{
  return(-1); // not implemented
}

int ApplyInverse( const Epetra_MultiVector & X,
      Epetra_MultiVector & Y ) const
{
  return(-1); // not implemented
}

double NormInf() const
{
  return(abs(diag_) + abs(diag_minus_one_) + abs(diag_plus_one_));
}

const char * Label () const
{
  return("TriDiagonalOperator");
}
```

```
  bool UseTranspose() const
  {
    return(false);
  }

  bool HasNormInf () const
  {
    return(true);
  }


  const Epetra_Comm & Comm() const
  {
    return(Map_.Comm());
  }

  const Epetra_Map & OperatorDomainMap() const
  {
    return(Map_);
  }

  const Epetra_Map & OperatorRangeMap() const
  {
    return(Map_);
  }


private:

  Epetra_Map Map_;
  double diag_minus_one_;   // value in the sub-diagonal
  double diag_;             // value in the diagonal
  double diag_plus_one_;    // value in the super-diagonal

};

// =========== //
// main driver //
// ----------- //

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  if( Comm.NumProc() != 1 ) {
    if( Comm.MyPID() == 0 ) {
      cerr << "This is mono-process example\n"
     << "Please run with one processo only\n";
    }
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    exit(EXIT_SUCCESS);
  }

  // global dimension of the problem, could be any positive number
  int NumGlobalElements( 5 );
```

```
  // linear decomposition (for simplicity, could be general)
  Epetra_Map Map(NumGlobalElements,0,Comm );

  // define two vectors based on Map
  Epetra_Vector x(Map);
  Epetra_Vector y(Map);
  x.PutScalar(1.0);

  // define a linear operator, as previously defined in class
  // TriDiagonalOperator

  TriDiagonalOperator TriDiagOp(-1.0,2.0,-1.0,Map);

  TriDiagOp.Apply(x,y);

  cout << x;
  cout << y;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 39

epetra_ex22

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Use of Epetra_Operator
// This code should be run with at least two processes

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Operator.h"
#include "Epetra_Import.h"
#include "Epetra_IntSerialDenseVector.h"

// auxiliary function to local an index in an integer vector

int find( int key, int vector[], int Length )
{
  for( int i=0 ; i<Length ; ++i ) {
    if( vector[i] == key ) return i;
  }
  return -1;
}

// ==================== //
// TriDiagonal Operator //
// -------------------- //
```

```
// NOTE: Will not work with IndexBase != 0
class TriDiagonalOperator : public Epetra_Operator
{

public:

  // constructor
  TriDiagonalOperator( double diag_minus_one,
           double diag,
           double diag_plus_one,
           const Epetra_Map & Map) :
    Map_( Map ),
    diag_minus_one_(diag_minus_one),
    diag_(diag),
    diag_plus_one_(diag_plus_one)
  {
    // build the importer
    // Each local node will need the node+1 and node-1
    // (except for global node 0 and global node NumGlobalElemenets-1
    NumMyElements_ = Map_.NumMyElements();
    NumGlobalElements_ = Map_.NumGlobalElements();
    int* MyGlobalElements = new int[NumMyElements_];
    Map_.MyGlobalElements(MyGlobalElements);

    // count the nodes required from other processors
    // (this will be an upper bound of the # of required nodes
    // because I may count twice an external node)
    int count=0;
    for( int i=0 ; i<NumMyElements_ ; ++i ) {
      int globalIndex = MyGlobalElements[i];
      // no -1 node for the first node of the grid
      if( globalIndex>0 )
  if( Map.LID(globalIndex-1) == -1 ) ++count;
      // now +1 node for the last node of the grid
      if( globalIndex<NumGlobalElements_-1 )
  if( Map.LID(globalIndex+1) == -1 ) ++count;
      ++count;
    }

    // now allocate space for local nodes and external nodes
    // (an external node is a node required for the matrix-vector
    // product, but owned by another process)
    int Length = count;
    int* ListOfNodes = new int[Length];

    count=0;
    for( int i=0 ; i<NumMyElements_ ; ++i ) {
      int globalIndex = MyGlobalElements[i];
      // no -1 node for the first node of the grid
      if( globalIndex>0 ) {
  if( Map.LID(globalIndex-1) == -1 )
    if( find( globalIndex-1, ListOfNodes, Length) == -1 ) {
      ListOfNodes[count] = globalIndex-1;
      ++count;
    }
      }
      // now +1 node for the last node of the grid
      if( globalIndex<NumGlobalElements_-1 ) {
  if( Map.LID(globalIndex+1) == -1 ) {
    if( find( globalIndex+1, ListOfNodes, Length) == -1 ) {
      ListOfNodes[count] = globalIndex+1;
```

```
    ++count;
  }
}
    }
    ListOfNodes[count] = globalIndex;
    ++count;
  }
  /*
  cout << "count = " << count << endl;
  for( int i=0 ; i<count ; i++ ) {
    cout << "ListOfNodes[" << i << "] = " << ListOfNodes[i] << endl;
  }
  */
  // create a Map defined using ListOfNodes

  ImportMap_ = new Epetra_Map(-1,count,ListOfNodes,0,Map_.Comm());

  Importer_ = new  Epetra_Import(*ImportMap_,Map_);

  delete[] MyGlobalElements;
  delete[] ListOfNodes;

  return;

}

// application of the tridiagonal operator
int Apply( const Epetra_MultiVector & X,
     Epetra_MultiVector & Y ) const
{

  cout << X;

  // maybe some error checks on MultiVector Lenghts
  // for the future...

  Epetra_MultiVector Xext((*ImportMap_),X.NumVectors());

  // this will contain local nodes and the required extenal nodes
  Xext.Import(X,*Importer_,Insert);

  for( int i=0 ; i<X.MyLength() ; ++i ) {

    int globalRow = Map_.GID(i);
    int iMinusOne = (*ImportMap_).LID(globalRow-1);
    int iPlusOne = (*ImportMap_).LID(globalRow+1);

    printf("%d %d %d\n", globalRow, iMinusOne, iPlusOne);

    for( int vec=0 ; vec<X.NumVectors() ; ++vec ) {

Y[vec][i] = diag_ * X[vec][i];

if( iMinusOne != -1 )
  Y[vec][i] += diag_minus_one_ * Xext[vec][iMinusOne];

if( iPlusOne != -1 )
  Y[vec][i] += diag_plus_one_ * Xext[vec][iPlusOne];

    }
  }
```

```
    return true;
  }

  // other function
  int SetUseTranspose( bool UseTranspose)
  {
    return(0);
  }

  int ApplyInverse( const Epetra_MultiVector & X,
        Epetra_MultiVector & Y ) const
  {
    return 0;
  }

  double NormInf() const
  {
    return( abs(diag_) + abs(diag_minus_one_) + abs(diag_plus_one_) );
  }

  const char * Label () const
  {
    return "TriDiagonalOperator";
  }

  bool UseTranspose() const
  {
    return false;
  }

  bool HasNormInf () const
  {
    return true;
  }


  const Epetra_Comm & Comm() const
  {
    return( Map_.Comm() );
  }

  const Epetra_Map & OperatorDomainMap() const
  {
    return( Map_ );
  }

  const Epetra_Map & OperatorRangeMap() const
  {
    return( Map_ );
  }


private:

  Epetra_Map Map_;
  int NumMyElements_;
  int NumGlobalElements_;
  double diag_minus_one_;   // value in the sub-diagonal
  double diag_;             // value in the diagonal
  double diag_plus_one_;    // value in the super-diagonal
  Epetra_Import *Importer_;
  Epetra_Map *ImportMap_;
```

```
};


// =========== //
// main driver //
// ----------- //

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // global dimension of the problem, could be any positive number
  int NumGlobalElements( 5 );

  // linear decomposition (for simplicity, could be general)
  Epetra_Map Map(NumGlobalElements,0,Comm );

  // define two vectors based on Map
  Epetra_Vector x(Map);
  Epetra_Vector y(Map);
  int NumMyElements = Map.NumMyElements();
  Epetra_IntSerialDenseVector MyGlobalElements(NumMyElements);
  Map.MyGlobalElements( MyGlobalElements.Values() );

  // x is a linear function, Laplace applied to it
  // should be zero except for the boundary nodes
  for( int i=0 ; i<NumMyElements ; ++i )
    x[i] = 1.0*MyGlobalElements[i];

  // define a linear operator, as previously defined in class
  // TriDiagonalOperator

  TriDiagonalOperator TriDiagOp(-1.0,2.0,-1.0,Map);

  TriDiagOp.Apply(x,y);

  cout << x;
  cout << y;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");
```

```
  return 0;
}
#endif
```

Chapter 40

epetra_ex23

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Use of Epetra_FECrsMatrix
// This code should be run with at least two processes

#include "Didasko_ConfigDefs.h"

#ifdef HAVE_DIDASKO_EPETRA

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_Vector.h"
#include "Epetra_FECrsMatrix.h"

// =========== //
// main driver //
// ----------- //

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // Create a linear map of size 10 (could be any number > 1)
  //
```

```
  // Note that a linear map is distributed approximately evenly over
  // all processors.
  //
  int NumGlobalElements = 10;
  Epetra_Map Map(NumGlobalElements,0,Comm);

  // create a diagonal FE crs matrix (one nonzero per row)
  Epetra_FECrsMatrix A(Copy,Map,1);

  // Next, set the matrix entries.
  //
  // Note 1: Matrix entries are only contributed from processor
  // 0, regardless of how many processors the program is running on.
  // Proc 0 fills the entire global matrix. Data that belongs in
  // matrix rows owned by procs 1 .. numprocs-1 gets sent to those
  // processors during the call to 'A.GlobalAssemble()' below.
  //
  // Note 2: We fill the matrix using 'InsertGlobalValues'. An
  // alternative approach that would be more efficient for large
  // matrices in most cases would be to first create and fill a
  // graph (Epetra_FECrsGraph), then construct the matrix with the
  // graph (after calling graph.FillComplete) and fill the matrix
  // using the method 'SumIntoGlobalValues'.
  //
  if( Comm.MyPID() == 0 ) {
    for( int i=0 ; i<NumGlobalElements ; ++i ) {
      int index = i;
      double value = 1.0*i;
      A.InsertGlobalValues(1,&index,&value);
    }
  }

  A.GlobalAssemble();

  cout << A;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra");

#ifdef HAVE_MPI
```

```
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 41

epetra_ex24

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Show the usage of RowMap, ColumnMap, RangeMap and DomainMap
// This code must be run with two processes

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Vector.h"
#include "Epetra_Map.h"
#include "Epetra_CrsMatrix.h"

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  int MyPID = Comm.MyPID();
  int NumProc = Comm.NumProc();

  if( NumProc != 2 ) {
#ifdef HAVE_MPI
    MPI_Finalize();
```

```
#endif
    return(EXIT_SUCCESS);
  }

  // I define two maps:
  // - MapA has 2 nodes on proc 0, 2 nodes on proc 1
  // - MapB has 4 nodes on proc 0, 0 nodes on proc 1

  int NumElementsA = 0, NumElementsB = 0;

  switch( MyPID ) {
  case 0:
    NumElementsA = 2;
    NumElementsB = 4;
    break;

  case 1:
    NumElementsA = 2;
    NumElementsB = 0;
    break;
  }

  Epetra_Map MapA(-1,NumElementsA,0,Comm);
  Epetra_Map MapB(-1,NumElementsB,0,Comm);

  // I create a diagonal matrix Q with RowMap,ColMap distibutions = MapA
  Epetra_CrsMatrix Q(Copy,MapA,MapA,1);

  int * MyGlobalElementsA = MapA.MyGlobalElements();

  // Now I put in Q the diagonal elements, using MapA
  for( int i=0 ; i<NumElementsA ; ++i ) {
    double one = 2.0;
    int indices = MyGlobalElementsA[i];
    Q.InsertGlobalValues(MyGlobalElementsA[i], 1, &one, &indices );
  }

  // Now, I would like to apply Q to a vector defined on
  // MapB, giving as a result a vector on MapB
  Q.FillComplete(MapB,MapB);

  // the instruction above causes the code to crash if
  // I build up the matrix as 'Epetra_CrsMatrix Q(Copy,MapA,1)'
  // (without specifing the col map)

  // create few vectors on the two maps
  Epetra_Vector VecA(MapA);   Epetra_Vector VecA2(MapA);
  Epetra_Vector VecB(MapB);   Epetra_Vector VecB2(MapB);

  VecA.PutScalar(1.0);        VecA2.PutScalar(1.0);
  VecB.PutScalar(1.0);        VecB2.PutScalar(1.0);

  Q.Multiply(false,VecB,VecB2);

  cout << VecB2;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);
```

```
}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra");

  return 0;
}
#endif
```

Chapter 42

epetra_ex25

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Basic definition of communicator.
// This code should be run with at least two processes

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_TRIUTILS)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Map.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Vector.h"
#include "Epetra_RowMatrix.h"
#include "Trilinos_Util.h"

class MSRMatrix : public Epetra_Operator
{

public:

  // constructor
  MSRMatrix(Epetra_Map Map, int * bindx, double * val) :
    bindx_(bindx),
    val_(val),
    Map_(Map)
  {
  }

  // destructor, nothing to do
```

```
~MSRMatrix()
{}

// Apply the RowMatrix to a MultiVector
int Apply(const Epetra_MultiVector & X, Epetra_MultiVector & Y ) const
{

  int Nrows = bindx_[0]-1;

  for( int i=0 ; i<Nrows ; i++ ) {
    // diagonal element
    for( int vec=0 ; vec<X.NumVectors() ; ++vec ) {
Y[vec][i] = val_[i]*X[vec][i];
    }
    // off-diagonal elements
    for( int j=bindx_[i] ; j<bindx_[i+1] ; j++ ) {
for( int vec=0 ; vec<X.NumVectors() ; ++vec ) {
  Y[vec][bindx_[j]] += val_[j]*X[vec][bindx_[j]];
}
    }
  }

  return 0;

} /* Apply */

// other function, required by Epetra_RowMatrix. Here are almost all
// void, you may decide to complete the example...
int SetUseTranspose( bool UseTranspose)
{
  return(-1); // not implemented
}

int ApplyInverse( const Epetra_MultiVector & X,
                  Epetra_MultiVector & Y ) const
{
  return(-1); // not implemented
}

double NormInf() const
{
  return -1;
}

const char* Label () const
{
  return "TriDiagonalOperator";
}

bool UseTranspose() const
{
  return false;
}

bool HasNormInf () const
{
  return true;
}


const Epetra_Comm & Comm() const
{
```

```
    return( Map_.Comm() );
  }

  const Epetra_Map & OperatorDomainMap() const
  {
    return( Map_ );
  }

  const Epetra_Map & OperatorRangeMap() const
  {
    return( Map_ );
  }

private:

  int * bindx_;    /* MSR vector for nonzero indices */
  double * val_;   /* MSR vector for nonzero values */
  Epetra_Map Map_;

}; /* MSRMatrix class */

// =========== //
// main driver //
// ----------- //

int main(int argc, char *argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  // define an Epetra communicator
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  // check number of processes
  if (Comm.NumProc() != 1) {
    if (Comm.MyPID() == 0)
      cerr << "*ERR* can be used only with one process" << endl;
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    exit(EXIT_SUCCESS);
  }

  // process 0 will read an HB matrix, and store it
  // in the MSR format given by the arrays bindx and val
  int N_global;
  int N_nonzeros;
  double * val = NULL;
  int * bindx = NULL;
  double * x = NULL, * b = NULL, * xexact = NULL;

  FILE* fp = fopen("../HBMatrices/fidap005.rua", "r");
  if (fp == 0)
  {
    cerr << "Matrix file not available" << endl;
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
    exit(EXIT_SUCCESS);
```

```
  }
  fclose(fp);

  Trilinos_Util_read_hb("../HBMatrices/fidap005.rua", 0,
      &N_global, &N_nonzeros,
      &val, &bindx,
      &x, &b, &xexact);

  // assign all the elements to process 0
  // (this code can run ONLY with one process, extensions to more
  // processes will require functions to handle update of ghost nodes)
  Epetra_Map Map(N_global,0,Comm);

  MSRMatrix A(Map,bindx,val);

  // define two vectors
  Epetra_Vector xxx(Map);
  Epetra_Vector yyy(Map);

  xxx.Random();

  A.Apply(xxx,yyy);

  cout << yyy;

  double norm2;
  yyy.Norm2(&norm2);

  cout << norm2 << endl;

  // free memory allocated by Trilinos_Util_read_hb
  if (val != NULL) free((void*)val);
  if (bindx != NULL) free((void*)bindx);
  if (x != NULL) free((void*)x);
  if (b != NULL) free((void*)x);
  if (xexact != NULL) free((void*)xexact);;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return(EXIT_SUCCESS);

} /* main */

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-triutils");

  return(0);
}
#endif
```

Chapter 43

epetraext_ex1

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

// Generate a matrix using triutils and redistribute with Zoltan.

#include "Didasko_config.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_EPETRAEXT) && defined(HA
      VE_ZOLTAN)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Trilinos_Util.h"
#include "Epetra_Comm.h"
#include "Epetra_Map.h"
#include "Epetra_Time.h"
#include "Epetra_BlockMap.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Vector.h"
#include "Epetra_Export.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_VbrMatrix.h"
#include "Epetra_LinearProblem.h"
#include "Trilinos_Util_CrsMatrixGallery.h"
#include "EpetraExt_Zoltan_CrsGraph.h"
// #include "EpetraExt_Transform.h"

int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm (MPI_COMM_WORLD);
```

```
#else
  Epetra_SerialComm Comm;
#endif

  int MyPID = Comm.MyPID();
  int n = 4; /* n by n grid */

  // Generate Laplacian2d gallery matrix
  Trilinos_Util::CrsMatrixGallery G("laplace_2d", Comm);
  G.Set("problem_size", n*n);
  G.Set("map_type", "linear"); // Linear map initially

  // Copy the CrsMatrix to A.
  Epetra_CrsMatrix A (*(G.GetMatrix()));
  // Epetra_CrsGraph Graph (A.Graph());

  cout << "Matrix A: " << A << endl;
  // cout << "Graph of A: " << A.Graph() << endl;

  // Repartition graph using Zoltan
  cout << "Calling Zoltan via EpetraExt to repartition the graph." << endl;
  // First create a transform
  EpetraExt::Zoltan_CrsGraph ZoltanTrans;
  // Then apply the transform to the graph.
  Epetra_CrsGraph & BalGraph = ZoltanTrans(const_cast<Epetra_CrsGraph&>(A.Graph()
      ));
  // Epetra_CrsGraph & BalGraph = ZoltanTrans(Graph);

  // cout << "Graph of A (not changed after balancing) : " << A.Graph() << endl;
  // cout << "Balanced Graph of A: " << BalGraph << endl;

  // Create Exporter to rebalance the matrix from the row maps
  Epetra_Export exporter(A.Graph().RowMap(), BalGraph.RowMap());

  cout << "Old rowmap for A: " << A.Graph().RowMap() << endl;
  cout << "New rowmap for A: " << BalGraph.RowMap() << endl;

  cout << "Proc " << MyPID << ": NumSend = " << exporter.NumSend()
       << ", NumRecv = " << exporter.NumRecv() << endl;

  Epetra_Time Timer(Comm);
  Comm.Barrier();
  double startTime = Timer.ElapsedTime();
  // Export A to new distribution.
  Epetra_CrsMatrix BalA(Copy, BalGraph);
  BalA.Export(A, exporter, Insert);
  Comm.Barrier();
  double matrixRedistributeTime = Timer.ElapsedTime() - startTime;

  cout << "Rebalanced matrix: " << BalA << endl ;
  if( MyPID==0 ) {
    cout << "Matrix redistribute time (sec) = "
   << matrixRedistributeTime << endl;
  }

#ifdef HAVE_MPI
  MPI_Finalize() ;
#endif

  return(0);
}
```

```
#else
#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-epetraext\n"
       "--enable-epetraext-zoltan");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 44

epetraext_ex2

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

// Generate a linearproblem using triutils and redistribute with Zoltan.
// This version uses transforms.
//
// *********** Under construction! Not working yet! **********************

#include "Didasko_config.h"
#if defined(HAVE_DIDASKO_EPETRA) && defined(HAVE_DIDASKO_EPETRAEXT) && defined(HA
      VE_ZOLTAN)

#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Trilinos_Util.h"
#include "Epetra_Comm.h"
#include "Epetra_Map.h"
#include "Epetra_Time.h"
#include "Epetra_BlockMap.h"
#include "Epetra_MultiVector.h"
#include "Epetra_Vector.h"
#include "Epetra_Export.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_VbrMatrix.h"
#include "Epetra_LinearProblem.h"
#include "Trilinos_Util_CrsMatrixGallery.h"
#include "EpetraExt_Zoltan_CrsGraph.h"
#include "EpetraExt_Transform.h"
#include "EpetraExt_LPTrans_From_GraphTrans.h"

int main(int argc, char *argv[]) {
```

```
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm (MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif

  int MyPID = Comm.MyPID();
  int n=4;

  // Epetra_Map  Map;
  // Epetra_CrsMatrix A;
  // Epetra_Vector x, b, xexact;

  // Generate Laplacian2d gallery matrix
  Trilinos_Util::CrsMatrixGallery G("laplace_2d", Comm);
  G.Set("problem_size", n*n);
  G.Set("map_type", "linear"); // Linear map initially
  //G.Set("exact_solution", "random");

  // Get the LinearProblem.
  Epetra_LinearProblem *Prob = G.GetLinearProblem();

  // Get the exact solution.
  Epetra_MultiVector *sol = G.GetExactSolution();

  // Get the rhs (b) and lhs (x)
  Epetra_MultiVector *b = Prob->GetRHS();
  Epetra_MultiVector *x = Prob->GetLHS();

  cout << "RHS b: " << *b << endl;
  cout << "LHS x: " << *x << endl;
  cout << "Solution: " << *sol << endl;

  // Repartition graph using Zoltan
  // Epetra_CrsGraph NewGraph = EpetraExt::Zoltan_CrsGraph(Graph);
  EpetraExt::Zoltan_CrsGraph * ZoltanTrans = new EpetraExt::Zoltan_CrsGraph();
  EpetraExt::LinearProblem_GraphTrans * ZoltanLPTrans =
    new EpetraExt::LinearProblem_GraphTrans(
         *(dynamic_cast<EpetraExt::StructuralSameTypeTransform<Epetra_CrsGraph>*>
      (ZoltanTrans)) );

  cout << "Creating Load Balanced Linear Problem\n";
  Epetra_LinearProblem &BalancedProb = (*ZoltanLPTrans)(*Prob);

  // Get the rhs (b) and lhs (x)
  Epetra_MultiVector *Balancedb = Prob->GetRHS();
  Epetra_MultiVector *Balancedx = Prob->GetLHS();
  cout << "Balanced b: " << *Balancedb << endl;
  cout << "Balanced x: " << *Balancedx << endl;


/****************** OLD STUFF not being used **********************

  int NumGlobalElements = readMap->NumGlobalElements();

  // Create uniform distributed map
  Epetra_Map map(NumGlobalElements, 0, Comm);

  // Create Exporter to distribute read-in matrix and vectors
```

```
  Epetra_Export exporter(*readMap, map);
  Epetra_CrsMatrix A(Copy, map, 0);
  Epetra_Vector x(map);
  Epetra_Vector b(map);
  Epetra_Vector xexact(map);

  Epetra_Time FillTimer(Comm);
  x.Export(*readx, exporter, Add);
  b.Export(*readb, exporter, Add);
  xexact.Export(*readxexact, exporter, Add);
  Comm.Barrier();
  double vectorRedistributeTime = FillTimer.ElapsedTime();
  A.Export(*readA, exporter, Add);
  Comm.Barrier();
  double matrixRedistributeTime = FillTimer.ElapsedTime() - vectorRedistributeTim
      e;
  assert(A.FillComplete()==0);
  Comm.Barrier();
  double fillCompleteTime = FillTimer.ElapsedTime() - matrixRedistributeTime;

  if( MyPID==0 ) {
    cout << "Vector redistribute  time (sec) = "
   << vectorRedistributeTime<< endl;
    cout << "Matrix redistribute time (sec) = "
   << matrixRedistributeTime << endl;
    cout << "Transform to Local  time (sec) = "
   << fillCompleteTime << endl<< endl;
  }

  delete readA;
  delete readx;
  delete readb;
  delete readxexact;
  delete readMap;

***********/

#ifdef HAVE_MPI
  MPI_Finalize() ;
#endif

  return(0);
}

#else
#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-epetra\n"
       "--enable-epetraext\n"
       "--enable-epetraext-zoltan");
```

```
#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#endif
```

Chapter 45

teuchos_ex1

```
// @HEADER
// ************************************************************************
//
//                     Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_TEUCHOS)
#ifdef HAVE_MPI
#include "mpi.h"
#endif

#include "Teuchos_SerialDenseMatrix.hpp"

int main(int argc, char* argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  // Creating a double-precision matrix can be done in several ways:
  // Create an empty matrix with no dimension
  Teuchos::SerialDenseMatrix<int,double> Empty_Matrix;
  // Create an empty 3x4 matrix
  Teuchos::SerialDenseMatrix<int,double> My_Matrix( 3, 4 );
  // Basic copy of My_Matrix
  Teuchos::SerialDenseMatrix<int,double> My_Copy1( My_Matrix ),
    // (Deep) Copy of principle 3x3 submatrix of My_Matrix
    My_Copy2( Teuchos::Copy, My_Matrix, 3, 3 ),
    // (Shallow) Copy of 2x3 submatrix of My_Matrix
    My_Copy3( Teuchos::View, My_Matrix, 2, 3, 1, 1 );

  // The matrix dimensions and strided storage information can be obtained:
  int rows, cols, stride;
  rows = My_Copy3.numRows();  // number of rows
  cols = My_Copy3.numCols();  // number of columns
  stride = My_Copy3.stride(); // storage stride

  // Matrices can change dimension:
```

```
  Empty_Matrix.shape( 3, 3 );      // size non-dimensional matrices
  My_Matrix.reshape( 3, 3 );       // resize matrices and save values

  // Filling matrices with numbers can be done in several ways:
  My_Matrix.random();             // random numbers
  My_Copy1.putScalar( 1.0 );      // every entry is 1.0
  My_Copy2(1,1) = 10.0;           // individual element access
  Empty_Matrix = My_Matrix;       // copy My_Matrix to Empty_Matrix

  // Basic matrix arithmetic can be performed:
  Teuchos::SerialDenseMatrix<int,double> My_Prod( 3, 2 );
  // Matrix multiplication ( My_Prod = 1.0*My_Matrix*My_Copy^T )
  My_Prod.multiply( Teuchos::NO_TRANS, Teuchos::TRANS,
        1.0, My_Matrix, My_Copy3, 0.0 );
  My_Copy2 += My_Matrix;         // Matrix addition
  My_Copy2.scale( 0.5 );         // Matrix scaling

  // The pointer to the array of matrix values can be obtained:
  double *My_Array, *My_Column;
  My_Array = My_Matrix.values();   // pointer to matrix values
  My_Column = My_Matrix[2];        // pointer to third column values

  // The norm of a matrix can be computed:
  double norm_one, norm_inf, norm_fro;
  norm_one = My_Matrix.normOne();        // one norm
  norm_inf = My_Matrix.normInf();        // infinity norm
  norm_fro = My_Matrix.normFrobenius();  // frobenius norm

  // Matrices can be compared:
  // Check if the matrices are equal in dimension and values
  if (Empty_Matrix == My_Matrix) {
    cout<< "The matrices are the same!" <<endl;
  }
  // Check if the matrices are different in dimension or values
  if (My_Copy2 != My_Matrix) {
    cout<< "The matrices are different!" <<endl;
  }

  // A matrix can be sent to the output stream:
  cout<< My_Matrix << endl;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#else

#include <stdlib.h>
#include <stdio.h>
#ifdef HAVE_MPI
#include "mpi.h"
#endif

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
```

```
       "--enable-teuchos");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 46

teuchos_ex2

```
// @HEADER
// ***********************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ***********************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#if defined(HAVE_DIDASKO_TEUCHOS)
#ifdef HAVE_MPI
#include "mpi.h"
#endif

#include "Teuchos_BLAS.hpp"

int main(int argc, char* argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  // Creating an instance of the BLAS class for double-precision kernels looks li
      ke:
  Teuchos::BLAS<int, double> blas;

  // This instance provides the access to all the BLAS kernels listed in Figure \
      ref{blas_kernels}:
  const int n = 10;
  double alpha = 2.0;
  double x[ n ];
  for ( int i=0; i<n; i++ ) { x[i] = i; }
  blas.SCAL( n, alpha, x, 1 );
  int max_idx = blas.IAMAX( n, x, 1 );
  cout<< "The index of the maximum magnitude entry of x[] is the "
      <<  max_idx <<"-th and x[ " << max_idx-1 << " ] = "<< x[max_idx-1]
      << endl;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
```

```
  return 0;
}
#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  puts("Please configure Didasko with:\n"
       "--enable-teuchos");

  return 0;
}
#endif
```

Chapter 47

teuchos_ex3

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#endif

#if defined(HAVE_DIDASKO_TEUCHOS)

#include "Teuchos_LAPACK.hpp"
#include "Teuchos_SerialDenseMatrix.hpp"
#include "Teuchos_SerialDenseVector.hpp"

int main(int argc, char* argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  // Creating an instance of the LAPACK class for double-precision routines looks
       like:
  Teuchos::LAPACK<int, double> lapack;

  // This instance provides the access to all the LAPACK routines.
  Teuchos::SerialDenseMatrix<int, double> My_Matrix(4,4);
  Teuchos::SerialDenseVector<int, double> My_Vector(4);
  My_Matrix.random();
  My_Vector.random();

  // Perform an LU factorization of this matrix.
  int ipiv[4], info;
  char TRANS = 'N';
  lapack.GETRF( 4, 4, My_Matrix.values(), My_Matrix.stride(), ipiv, &info );

  // Solve the linear system.
```

```
  lapack.GETRS( TRANS, 4, 1, My_Matrix.values(), My_Matrix.stride(),
    ipiv, My_Vector.values(), My_Vector.stride(), &info );

  // Print out the solution.
  cout << My_Vector << endl;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-teuchos");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 48

teuchos_ex4

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#endif
#if defined(HAVE_DIDASKO_TEUCHOS)

#include "Teuchos_ParameterList.hpp"
#include "Teuchos_ConfigDefs.hpp"

int main(int argc, char* argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  // Creating an empty parameter list looks like:
  Teuchos::ParameterList My_List;

  // Setting parameters in this list can be easily done:
  My_List.set("Max Iters", 1550);
  My_List.set("Tolerance", 1e-10);
  My_List.set("Solver", "GMRES");

  /* The templated ``set'' method should cast the input {\it value} to the
     correct data type.  However, in the case where the compiler is not casting t
      he input
     value to the expected data type, an explicit cast can be used with the ``set
      '' method:
  */
  My_List.set("Tolerance", (float)(1e-10));

  /* A hierarchy of parameter lists can be constructed using {\tt Teuchos::Parame
      terList}.  This
```

```
   means another parameter list is a valid {\it value} in any parameter list.
    To create a sublist
   in a parameter list and obtain a reference to it:
*/
Teuchos::ParameterList& Prec_List = My_List.sublist("Preconditioner");

// Now this parameter list can be filled with values:
Prec_List.set("Type", "ILU");
Prec_List.set("Drop Tolerance", 1e-3);

// The parameter list can be queried about the existance of a parameter, sublis
    t, or type:
// Has a solver been chosen?
bool solver_defined, prec_defined, tol_double, dtol_double;
solver_defined = My_List.isParameter("Solver");
// Has a preconditioner been chosen?
prec_defined = My_List.isSublist("Preconditioner");
// Has a tolerance been chosen and is it a double-precision number?
tol_double = My_List.INVALID_TEMPLATE_QUALIFIER isType<double>("Tolerance");
// Has a drop tolerance been chosen and is it a double-precision number?
dtol_double = Teuchos::isParameterType<double>(Prec_List, "Drop Tolerance");

/* The last two methods for checking the parameter type are equivalent.
   There is some question as to whether the syntax of the first type-checking
   method is acceptable to older compilers.  Thus, the second type-checking met
    hod
   is offered as a portable alternative.
*/
// Parameters can be retrieved from the parameter list in quite a few ways:
// Get method that creates and sets the parameter if it doesn't exist.
int its;
its = My_List.get("Max Iters", 1200);
// Get method that retrieves a parameter of a particular type.
float tol;
tol = My_List.INVALID_TEMPLATE_QUALIFIER get<float>("Tolerance");

/* In the above example, the first ``get'' method is a safe way of
   obtaining a parameter when its existence is indefinite but required.
   The second ``get'' method should be used when the existense of the parameter

   is definite.  This method will throw an exception if the parameter doesn't e
    xist.
   The safest way to use the second ``get'' method
   is in a try/catch block:
*/
try {
  tol = My_List.INVALID_TEMPLATE_QUALIFIER get<float>("Tolerance");
}
catch (std::exception& e) {
  tol = 1e-6;
}

/* The second ``get'' method uses a syntax that may not be
   acceptable to older compilers.  Optionally, there is another portable templa
    ted
   ``get'' function that can be used in the place of the second ``get'' method:

*/
try {
  tol = Teuchos::getParameter<float>(My_List, "Tolerance");
}
catch (std::exception& e) {
```

```
    tol = 1e-6;
  }

  // A parameter list can be sent to the output stream:
  cout<< My_List << endl;

  /* It is important to note that mispelled parameters
     (with additional space characters, capitalizations, etc.) may be ignored.
     Therefore, it is important to be aware that a given parameter has not been u
      sed.
     Unused parameters can be printed with method:
  */
  My_List.unused( cout );

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-teuchos");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 49

teuchos_ex5

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#endif
#if defined(HAVE_DIDASKO_TEUCHOS)

#include "Teuchos_RCP.hpp"

class A {
 public:
   A() {}
   virtual ~A(){}
   virtual void f(){}
};
class B1 : virtual public A {};
class B2 : virtual public A {};
class C : public B1, public B2 {};

using namespace Teuchos;

int main(int argc, char* argv[])
{

#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  // Create some reference-counted pointers.
  // Create a reference-counted NULL pointer of type A.
  RCP<A>             a_null_ptr;
  // Create a reference-counted pointer of non-const type A.
  RCP<A>             a_ptr = rcp(new A);
  // Create a reference-counted pointer of const type A.
```

```
  RCP<const A>       ca_ptr = rcp(new A);
  // Create a const reference-counted pointer of non-const type A.
  const RCP<A>       a_cptr = rcp(new A);
  // Create a const reference-counted pointer of const type A.
  const RCP<const A> ca_cptr = rcp(new A);

  // Perform implicit conversions between a derived class and its base class.
  RCP<B1> b1_ptr  = rcp(new B1);
  RCP<A> a_ptr1 = b1_ptr;

  /* Other non-implicit type conversions like static, dynamic, or const casts
     can be taken care of by non-member template functions.
  */
  RCP<const C> c_ptr = rcp(new C);
  // Implicit cast from C to B2.
  RCP<const B2> b2_ptr = c_ptr;
  // Safe cast, type-checked, from C to A.
  RCP<const A> ca_ptr1 = rcp_dynamic_cast<const A>(c_ptr);
  // Unsafe cast, non-type-checked, from C to A.
  RCP<const A> ca_ptr2 = rcp_static_cast<const A>(c_ptr);
  // Cast away const from B2.
  RCP<B2>       nc_b2_ptr = rcp_const_cast<B2>(b2_ptr);

  /* Using a reference-counted pointer is very similar to using a raw C++ pointer
      .  Some
     of the operations that are common to both are:
  */
  RCP<A>
    a_ptr2 = rcp(new A), // Initialize reference-counted pointers.
    a_ptr3 = rcp(new A); // ""
  A  *ra_ptr2 = new A,    // Initialize non-reference counted pointers.
    *ra_ptr3 = new A;    // ""
  a_ptr2 = rcp(ra_ptr3);  // Assign from a raw pointer (only do this once!)
  a_ptr3 = a_ptr1;        // Assign one smart pointer to another.
  a_ptr2 = rcp(ra_ptr2);  // Assign from a raw pointer (only do this once!)
  a_ptr2->f();            // Access a member of A using ->
  ra_ptr2->f();           // ""
  *a_ptr2 = *a_ptr3;      // Dereference the objects and assign.
  *ra_ptr2 = *ra_ptr3;    // ""

  // Get the raw C++ pointer.
  A* true_ptr = 0;
  true_ptr = a_ptr1.get();

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
```

```
       "--enable-teuchos");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 50

teuchos_ex6

```
// @HEADER
// ************************************************************************
//
//                    Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#endif
#if defined(HAVE_DIDASKO_TEUCHOS)

#include "Teuchos_TimeMonitor.hpp"

using namespace Teuchos;

// Global Timers
RCP<Time> CompTime = TimeMonitor::getNewTimer("Computational Time");
RCP<Time> FactTime = TimeMonitor::getNewTimer("Factorial Time");

// Quadratic function declaration.
double quadFunc( double x );

// Factorial function declaration.
double factFunc( int x );

int main(int argc, char* argv[])
{
  int i;
  double x;

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
#endif

  // Apply the quadratic function.
  for( i=-100; i<100; i++ ) {
    x = quadFunc( (double) i );
  }
```

```
  // Apply the factorial function.
  for( i=0; i<100; i++ ) {
    x = factFunc( i );
  }

  // Get a summary from the time monitor.
  TimeMonitor::summarize();

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return 0;
}

/* Evaluate a quadratic function at point x */
double quadFunc( double x )
{
  // Construct a local time monitor, this starts the CompTime timer and will stop
       when leaving scope.
  Teuchos::TimeMonitor LocalTimer(*CompTime);

  // Evaluate the quadratic function.
  return ( x*x - 1.0 );
}

/* Compute the factorial of x */
double factFunc( int x )
{
  // Construct a local time monitor, this starts the FactTime timer and will stop
       when leaving scope.
  Teuchos::TimeMonitor LocalTimer(*FactTime);

  // Special returns for specific cases.
  if( x == 0 ) return 0.0;
  if( x == 1 ) return 1.0;

  // Evaluate the factorial function.
  return ( (double) x * factFunc(x-1) );
}

#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:\n"
       "--enable-teuchos");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 51

teuchos_ex7

```
// @HEADER
// ************************************************************************
//
//                     Didasko Tutorial Package
//                Copyright (2005) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
//
// Questions about Didasko? Contact Marzio Sala (marzio.sala _AT_ gmail.com)
//
// ************************************************************************
// @HEADER

#include "Didasko_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#endif
#if defined(HAVE_DIDASKO_TEUCHOS)

#include "Teuchos_CommandLineProcessor.hpp"

int main(int argc, char* argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  // Creating an empty command line processor looks like:
  Teuchos::CommandLineProcessor My_CLP;

  /* To set and option, it must be given a name and default value.  Additionally,

     each option can be given a help string.  Although it is not necessary, a hel
      p
     string aids a users comprehension of the acceptable command line arguments.
     Some examples of setting command line options are:
  */
  // Set an integer command line option.
  int NumIters = 1550;
  My_CLP.setOption("iterations", &NumIters, "Number of iterations");
  // Set a double-precision command line option.
  double Tolerance = 1e-10;
  My_CLP.setOption("tolerance", &Tolerance, "Tolerance");
  // Set a string command line option.
  string Solver = "GMRES";
  My_CLP.setOption("solver", &Solver, "Linear solver");
```

```
  // Set a boolean command line option.
  bool Precondition;
  My_CLP.setOption("precondition","no-precondition",
       &Precondition,"Preconditioning flag");

  /* There are also two methods that control the strictness of the command line p
      rocessor.
     For a command line processor to be sensitive to any bad command line option
      that it
     does not recognize use:
  */
  My_CLP.recogniseAllOptions(false);

  /* Then, if the parser finds a command line option it doesn't recognize, it wil
      l
     throw an exception.  To prevent a command line processor from throwing an ex
      ception
     when it encounters a unrecognized option or help is printed, use:
  */
  My_CLP.throwExceptions(false);

  //Finally, to parse the command line, argc and argv are passed to the parse met
      hod:
  My_CLP.parse( argc, argv );

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#else

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
#endif

  puts("Please configure Didasko with:");
  puts("--enable-teuchos");

#ifdef HAVE_MPI
  MPI_Finalize();
#endif
  return 0;
}
#endif
```

Chapter 52

File Index

52.1 File List

Here is a list of all files with brief descriptions:

Chapter 53

File Documentation

53.1 mainpage.doxygen File Reference

Index