SANDIA REPORT

SAND20XX-????
Unlimited Release
Printed ??

Albany Development: Getting Started

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND20XX-????
Unlimited Release
Printed ??

Albany Development: Getting Started

Abstract

This document is intended to help new developers get started contributing to Albany. Albany is the main demonstration application of the AgileComponets strategy. It is a PDE code that strives to be built almost entirely from functionality in reusable libraries (such as Trilinos/STK/Dakota). Albany plays a large role in demonstrating and maturing functionality of new libraries, and also in the interfaces and interoperability between these libraries. It also serves to expose gaps in coverage of capabilities and interface. Another aspect of the project is to serve as a template for writing new applications against Trilinos/STK/Dakota. It uses CMake, CTest, and git, and grabs almost all configuration information from installed Trilinos. Albany was granted OpenSource license to share with collaborators.

Albany is also the home for several application and algorithm projects that use and contribute to the overall infrastructure. These include the LCM (Laboratory for Computatinal Mechanics), QCAD (Quantum Computer Aided Design) and FELIX (Finite Element for Land Ice eXperiments) applications, as well as algorthmic projects in embedded uncertainty quantification, model order reduction, and maturation of the templated software stack in Trilinos.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Albany is the main demonstration application of the AgileComponets strategy. It is a PDE code that strives to be built almost entirely from functionality in reusable libraries (such as Trilinos/STK/Dakota). Albany plays a large role in demonstrating and maturing functionality of new libraries, and also in the interfaces and interoperability between these libraries. It also serves to expose gaps in our coverage of capabilities and interface. Another aspect of the project is to serve as a template for writing new applications against Trilinos/STK/Dakota. It uses CMake, CTest, and git, and grabs almost all configuration information from installed Trilinos. Albany was granted OpenSource license to share with collaborators.

Albany is also the home for several application and algorithm projects that use and contribute to the overall infrastructure. These include the LCM (Laboratory for Computatinal Mechanics), QCAD (Quantum Computer Aided Design) and FELIX (Finite Element for Land Ice eXperiments) applications, as well as algorthmic projects in embedded uncertainty quantification, model order reduction, and maturation of the templated software stack in Trilinos.

Distinguishing Capabilities

The highlight of Albany is the PDE assembly. The template-based generic programming approach allows developers to just program for residual equations, and all manner of derivatives and polynomial propogations get automatically computed with no development effort. This approach uses Phalanx for rapid and flexible addition of physics, which works closely with Sacado and Stokhos for automatic propagation of derivatives and UQ. Intrepid and Shards packages are used for the local discretization.

A second strength of Albany is the demonstration of transformational analysis algorithms. Albany demonstrates the clean use of all Solver/Analysis tools in Trilinosi (through Piro, which was developed in Albany) including NOX, LOCA, Rythmos, Stokhos, and all of Dakota. On any problem we not only get a solution, but can also get sensitivities, run optimization problems, and perform uncertainty quantification. All of these approaches can access all of the linear solver options in Trilinos that are exposed by the Stratimikos layer.

The third main strength is the early adoption of STK, the sierra toolkit libraries. This includes the mesh database, IO, and will be growing soon to include mesh changes for stk_rebalance and stk_adapt.

Also of note is that Albany is, and intends to remain, a Publically Available code base, free of export control restrictions. This allows it to serve freely as a collaboration vehicle with universities and other labs, and as a template for building applications from Trilinos.

Physics Sets: what PDEs

Albany strives for a bit of a paradigm shift, where a code is defined more by its analysis capabilities and data structure choices, which are difficult to change, and less by the physics set. Much of Albany was developed in FY08-10 for solving simple heat transfer and poisson equations. With nonlinear source terms, this was adequate for developing and verifying all the hooks for analysis algorithms from sensitivity analysis to UQ. Recenty, Navier-Stokes equations have been added to the general Albany physics set. In FY11, two new projects (LCM and QCAD) were funded to develop application codes on Albany. These physics sets are developed in the Albany code, yet are distinct in many ways as well: C++ namespace, project teams, and funding sources. More recenlty, other applications and algorithm research projects have chose Albany as their home.

LCM: Labratory for Computational Mechanics

The first application project build on Albany is the LCM, or Laboratory for Computational Mechanics [Ostien-PI, Salinger, Mota, Foulk, Littlewood]. This project is creating an OpenSource computational mechanics RandD code, with a particular emphasis on fracture and failure models. The infrastructure in Albany, which is aimed at rapid development of new physics with automated generation of analytic derivatives and sensitivities, is well suited for trying out new discretizations and new material models. The links to Dakota and handling of model parameter are set up to perform calibration and UQ studies. LCM serves as a research tool that can be shared with academics. Successful research ideas and code will be migrated to the production mechanics applications of Adagio and Presto. Albany can link against the LAME material library, and we are looking at ways of getting derivative info through it for more robust solution algorithms.

Mechanics and Multi-Physics

Many engineering applications are multi-physical, in which cases mechanical behaviors are largely depending on both the constitutive responses and other physical processes taking

place within the solid. To accurately replicate and predict multiphysical phenomena, additional constraint(s) must be added to the governing equations such that all physical processes are represented properly. LCM is equipped with a number of mixed finite element models aimed to capture the following multiphysical phenomena: thermomechanics, poromechanics, and hydrogen diffusion-deformation problems.

As the name implied, the thermomechanics problem deals with standard dissipative solids under non-isothermal condition. In such a condition, deformation may generate heat while heat diffusion may occur simultaneously within the body. In LCM, the thermomechanical process is modeled by a multiplicative decomposition of deformation gradient which takes account of the deformation induced by thermal expansion, elastic energy storage and plastic dissipation. The balance of linear momentum is coupled with the balance of energy equation to capture the coupling between the solid deformation and thermal diffusion processes.

Poromechanics problem concerns with porous solids infiltrated with liquid or gas. Examples of porous solids include bone, soft tissue, sand, clay, rock and concrete. The hydraulic-mechanical coupling is two-way. On the one hand, deformation may trigger seepage within the porous media. On the other hand, hydraulic response may also introduce time-dependence and gradient-dependence into the mechanical behavior as the transient diffusion takes place. In the case where pore-fluid is trapped inside the host matrix, coupling between the solid and fluid constituents may lead to isochoric deformation. In LCM, this coupled physics is modeled via equal-order finite element spaces for displacement and pore pressure. Since this discretization choice may lead to spurious oscillation, an adaptive pressure projection stabilization scheme is introduced to guarantee stable numerical solutions in both infinitesimal and finite deformation regimes. A concurrent coupling is used such that a phenomenon called Mandel-Cryer effect can be properly captured.

In addition, LCM is also capable of modeling hydrogen transport within metals. This model is used to analyze how presence of hydrogen affects fracture of metals in various concentration levels. Since hydrogen-gas is increasingly popular to be used as an energy source, hydrogen embrittlement of metals is a key material issues one needed to address for future energy security. Mathematically, hydrogen transport can be modeled as a nonlinear convection-diffusion problem, in which the dislocation density may act as a source term that affects the distribution of hydrogen between the lattice site and the trapped site. To avoid spurious solutions commonly exhibited in convection-diffusion problem at low diffusivity limit, we introduce an adaptively, projection based stabilization scheme, which may reduce to lumped and higher-order mass formulation in one dimension.

QCAD: Quantum Device Design

The other new application code built on Albany is the QCAD quantum dot design LDRD (Schrodinger-Poisson) [Muller-PI, Gao, Nielsen, Salinger]. A poisson solve for classical representation of charge distribution is coupled to a Schrodinger region for quantum effects.

Uncertainity Quantification (UQ)

Albany is also serving as a main development and demonstration environment for embedded UQ research [Phipps-PI, Wildey]. By leveraging the templated fill environment, polynomial representations can be directly propogated through the physics assembly. Many issues with data structures, parallelization, and linear algebra are being addressed.

NEAMS

FELIX

A new project, as of late FY12, using Albany as a PDE solver is the PISCEES SciDAC-3 project joint between the BER ans ASCR divisions of the office of science. This project is developing simulation tools for Ice Sheet dynamics (targeting Greenland and then Antarctica). The initial models going into the FELIX (Finite Element Land Ice Experiments) namespace within Albany area a 3D Stokes model with nonlinear viscosity formulation plus $2-3$ other models that are simplifications of Stokes. This project will drive much development in UQ through interactions with Dakota. One unique feature of this project is that it will be able to use a mesh from LANL's MPAS framework, putting to test the abstraction layer that we put between the finite element asembly and the concrete STK Mesh implementation. Long term plans include adjoint for distributed boundary condition parameters, coupled temperature solves, and implicit advection of the Ice Sheet. The plan is for this code to be integrated into the CESM Community Earth System Model at NCAR as an option for the CISM Community Ice Sheet Model.

Nuclear Energy Reactor

Further development

The new applications have exposed many weaknesses in the Albany code, and more importantly, some gaps in the aggregate Agile Components infrastructure. For instance, an initial implementation in Trilinos of time integrators has been developed in responses to the needs of the transient dynamics problem. Future development includes: load balancing and uniform mesh refinement, transitioning to Tpetra and a templated software stack, being a testbed for embedded UQ methods, early adoption of architecture-aware PDE assembly

kernels, and eventually a full error estimation and adaptivity capability. Albany is making the transition from a demonstration prototype to a research code. It still seriously lacks full boundary condition support, any multi-physics capability, post-processing, and documentation (all the hard stuff). The sister code Drekar [Pawlowski, Cyr] has developed the infrastructure for multi-physics applications with varying physics and discretizations, which Albany does not support.

Chapter 2

Building

Building Albany, at a minimum, requires nothing but an installation of Trilinos. The AgileComponents philosophy is geared toward usage of multiple packages from the Trilinos suite of codes. Therefore, task number one is the acquisition, build, and installation of Trilinos. This chapter will layout the requirements for building the necessary third party libraries, as well as building and installing the proper Trilinos packages.

It is assumed that at this point you have a copy of the Albany code, possibly via a clone of the git repository. If not, see the beginning of Chapter 4 to see how to use `git` to get development versions of Albany and Trilinos. These commands require an account on the machine `software.sandia.gov` and being a member of the `trilinos` UNIX group.

In the `Albany/doc` directory there are some example script that provide templates for various steps of the build process. In particular, the configuration stage for Trilinos requires a CMake script, and an example can be found in the `trilinos-cmake` file, which will be reproduced here for completeness. Typical usage is to make a "build" directory under Trilinos and copy the `trilinos-cmake` file to the build directory, and edit any machine-specific paths. The last line of this script, `../`, is the relative path from the build directory to the main `Trilins` directory.

Example cmake file to configure Trilinos

```
#
# This is a sample Trilinos configuration script for Albany.
#
# Boost is required, but just needs to be unpacked,
# not compiled. Version _1_40 or newer.
#
# There are two optional build choices, commented below
#   these are for Dakota and Exodus I/O capabilites.
#
# Albany automatically queries the Trilinos build to
# know if these capabilities are enabled or disabled.
```

```
#
#
# All paths must me changed for your build (search "agsalin")
#
rm CMakeCache.txt
PREFIX=$PWD/install
BOOSTDIR=/home/agsalin/install/boost_1_49_0

cmake -D CMAKE_INSTALL_PREFIX:PATH=$PREFIX \
      -D Boost_INCLUDE_DIRS:FILEPATH=$BOOSTDIR \
      -D CMAKE_BUILD_TYPE:STRING=NONE \
      -D Trilinos_WARNINGS_AS_ERRORS_FLAGS:STRING="" \
      -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \
      -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=OFF \
\
      -D Trilinos_ENABLE_Teuchos:BOOL=ON \
      -D Trilinos_ENABLE_Shards:BOOL=ON \
      -D Trilinos_ENABLE_Sacado:BOOL=ON \
      -D Trilinos_ENABLE_Epetra:BOOL=ON \
      -D Trilinos_ENABLE_EpetraExt:BOOL=ON \
      -D Trilinos_ENABLE_Ifpack:BOOL=ON \
      -D Trilinos_ENABLE_AztecOO:BOOL=ON \
      -D Trilinos_ENABLE_Amesos:BOOL=ON \
      -D Trilinos_ENABLE_Anasazi:BOOL=ON \
      -D Trilinos_ENABLE_Belos:BOOL=ON \
      -D Trilinos_ENABLE_ML:BOOL=ON \
      -D Trilinos_ENABLE_Phalanx:BOOL=ON \
      -D Trilinos_ENABLE_Intrepid:BOOL=ON \
      -D Trilinos_ENABLE_NOX:BOOL=ON \
      -D Trilinos_ENABLE_Stratimikos:BOOL=ON \
      -D Trilinos_ENABLE_Thyra:BOOL=ON \
      -D Trilinos_ENABLE_Rythmos:BOOL=ON \
      -D Trilinos_ENABLE_MOOCHO:BOOL=ON \
      -D Trilinos_ENABLE_OptiPack:BOOL=ON \
      -D Trilinos_ENABLE_GlobiPack:BOOL=ON \
      -D Trilinos_ENABLE_Stokhos:BOOL=ON \
      -D Trilinos_ENABLE_Isorropia:BOOL=ON\
      -D Trilinos_ENABLE_Piro:BOOL=ON \
      -D Trilinos_ENABLE_STK:BOOL=ON \
      -D Trilinos_ENABLE_Teko:BOOL=ON \
      -D Trilinos_ENABLE_Zoltan:BOOL=ON \
\
      -D Trilinos_ENABLE_Mesquite:BOOL=OFF\
      -D Trilinos_ENABLE_Zoltan:BOOL=ON\
      -D Trilinos_ENABLE_FEI:BOOL=OFF\
```

```
\
      -D Trilinos_ENABLE_TESTS:BOOL=OFF \
      -D Piro_ENABLE_TESTS:BOOL=ON \
      -D Trilinos_ENABLE_EXAMPLES:BOOL=OFF \
      -D TPL_ENABLE_MPI:BOOL=ON \
      -D TPL_ENABLE_Boost:BOOL=ON \
\
      -D Phalanx_ENABLE_TEUCHOS_TIME_MONITOR:BOOL=ON \
      -D Stokhos_ENABLE_TEUCHOS_TIME_MONITOR:BOOL=ON \
      -D Stratimikos_ENABLE_TEUCHOS_TIME_MONITOR:BOOL=ON \
\
      -D CMAKE_VERBOSE_MAKEFILE:BOOL=OFF \
      -D Trilinos_VERBOSE_CONFIGURE:BOOL=OFF \
      -D CMAKE_CXX_FLAGS:STRING="-g -O2 -fno-var-tracking" \
      -D Trilinos_ENABLE_Export_Makefiles:BOOL=ON \
       ../

#
# Optional build capabilities:
# (1) TriKota is a Trilinos package that builds the
#     Dakota libraries, for optimization and UQ. See
#     TriKota web page for how to unpack Dakota.
#     Dakota requires boost libraries. See boost-make
#     sample script for how to build these libraries.
#
#        -D Trilinos_ENABLE_TriKota:BOOL=ON \
#        -D TriKota_ENABLE_DakotaCMake:BOOL=ON \
#        -D DAKOTA_ENABLE_TESTS:BOOL=OFF \
#        -D Boost_LIBRARY_DIRS:FILEPATH="$BOOSTDIR/lib" \
#
#
# (2) These 6 lines regarding  SEACAS/netcdf  are needed
#     for reading exodus meshes, but require an
#     installed netcdf. Also used for Pamgen meshes.
#        -D Trilinos_ENABLE_SEACASIoss:BOOL=ON \
#        -D Trilinos_ENABLE_Pamgen:BOOL=ON \
#        -D TPL_ENABLE_Netcdf:BOOL=ON \
#        -D SEACASExodus_ENABLE_MPI:BOOL=OFF \
#        -D TPL_Netcdf_INCLUDE_DIRS:PATH=/home/agsalin/install/netcdf-4.0.1/include \
#        -D Netcdf_LIBRARY_DIRS:PATH=/home/agsalin/install/netcdf-4.0.1/lib \
```

After executing this script, it should suffice to do

```
% make && ctest && make install
```

where the enabled tests should pass. In the above script, this is just a handful of `piro` tests which are a good indicator if the build was successful.

For runs using the exodus mesh database, which is the majority of examples and applications, it is necessary to have a set of the SEACAS tools installed on your machine. These can now be built from Trilinos. We recommend a separate build for these (as opposed to doing them as part of the Trilinos install above). The file `Albany/doc/seacas-cmake` is a script that will configure trilinos to build these tools. Again, this should be followed by a `make; make install` and making sure the executables, e.g. `epu`, are in your path.

Third Party Libraries

To build Trilinos with the aim of building Albany, a number of software dependencies must be met. They are termed third party libraries (TPLs).

1. Git
2. A recent version of Boost
3. NetCDF - version $\geq$ 4.1.3 (version $>$ 4.2 also requires HDF5)
4. CMake - version $>$ 2.7 should be fine
5. Some version of MPI, possibly openmpi - version $\geq$ 1.4.3
6. BLAS/LAPACK - systeminstalled version on relatively modern Linux/Mac machine works
7. Optional - a recent version (4.7) of GCC

There are some other products that can aid workflow.

1. eg – Easy Git, makes some things clearer and cleaner
2. doxygen/graphviz and dot - to build the doxygen documentation and visualize the phalanx graphs
3. paraview to post-process
4. cubit for mesh generation (not currently free, for now)
5. Optional - php (a php server is required if you want a local build of the Albany website, useful for visualizing documentation, but not necessary)

Example CMake File for Albany

The following cmake configuration script is enough to configure Albany.

```
#!/bin/bash
rm CMakeCache.txt
cmake  \
    -D ALBANY_TRILINOS_DIR:FILEPATH=<location_of_trilinos_install> \
    ../
```

Note that the <location_of_trilinos_install> needs to be exacly the path in the CMAKE_INSTALL_PREFIX in the Trilinos build above. This is typically executed from a subdirectory of Albany such as build or build_linux_mpi_20120920 depending on your personal directory-naming style. Note, that the final ../ is the relative path to the Albany directory from the build directory where this script is invoked.

After invoking the script, it remains to build Albany and run the tests, which can be accomplished from within the build directory using the following command.

```
% make -j 4 && ctest
```

If everyting is well, all the tests should pass.

Numerous parts of the build process are taken from the Trilinos install. For instance, the compilers, compiler flags, and any paths to netcdf, boost, blas, lapack, etc., are taken from the Trilinos build and do not need to be repeated in the Albany configuration.

In addition, the Albany build system will auto-detect what packages were built in Trilinos and set defines in Albany to accommodate. For example, the presence of the Zoltan package in the Trilinos install will trigger the definition of ALBANY_ZOLTAN on compile lines. Corresponding #ifdef ALBANY_ZOLTAN lines in the source code will toggle the compilation of capabilities in Albany that require Zoltan, such as stk_rebalance. The same is true for Dakota, SEACASIoss (for reading exodus files), and MPI (versus serial).

There are other CMake configuration options recognized by the Albany build system. (In addition, CMake has a standard set that can be found at the CMake websote.) Many of these are experimental options and not generally supported, but currrently include [default]:

```
ENABLE_DEMO_PDES       & Bool flag to enable Albany PDES such as Navier-Stokes [on] \\
ENABLE_LCM             & Bool flag to enable LCM physics sets [off] \\
ENABLE_QCAD            & Bool flag to enable QCAD physics sets [on] \\
ENABLE_LANDICE           & Bool flag to enable FELIX physics sets [off] \\
ENABLE_HYDRIDE         & Bool flag to enable Hydride physics set [off] \\
\\
```

```
ENABLE_MOR             & Bool flag to enable MOR Model Order Reduction code [on] \\
ENABLE_ASCR            & Bool flag to enable ASCR-funded embedded system UQ research [of
ENABLE_LAME            & Bool flag to enable links to the LAME material library [off] \\
ENABLE_LAMENT          & Bool flag to enable links to the LAMENT material library [off]
ENABLE_SCOREC          & Bool flag to enable links to RPI/SCOREC's FMBD mesh library [of
LAME_INCLUDE_DIR       & Path to installed Lame include directory \\
LAME_LIBRARY_DIR       & Path to installed Lame lib directory \\
ENABLE_ALBANY_CI       & Flag to enable links to the CI configuration interaction librar
ALBANY_CI_INCLUDE_DIR  & Path to installed CI include directory \\
ALBANY_CI_LIBRARY_DIR  & Path to installed CI lib directory \\
ALBANY_CXX_FLAGS       & Extra flags for the C++ compiler appended to those from Trilino
CMAKE_CXX_FLAGS        & Flags for the C++ compiler overwriting those from Trilinos \\
CMAKE_VERBOSE_MAKEFILE & Option to turn on verbose makefiles with full compile and link
```

The Heat transfer Problem in Albany is always enabled, as well as the many tests that use this simple physics set. All other PDEs (physics sets) can be toggled by the first several configuration options in the above list (before the blank line).

Chapter 3

Albany Directory Structure

This Chapter is meant to orient a new developer to the directory structure of the Albany repository. At the top level, Albany has three directories: `src` for the source code, `examples` for the example problems (which also serve as the regression tests), and a `doc` documentation directory.

Source Code Directories `Albany/src`

- `Albany/src` This top-level source code directory `src` contains much of the generic Albany code that is application independent. This includes the top-level functionality of `Main` routines and the factories for building the Trilinos `piro` solvers. The interface to the physics sets is the `Albany_ModelEvaluator`, a concrete implementation of a central abstract layer in Trilinos.

  Perhaps the most central piece of code is the `Albany_Application` class. The constructor of this class orchestrates the building of the discretization (mesh), the problem (physics set), initial guess, and responses. The methods in this class are called directly from the `Albany_ModelEvaluator` and compute the residual, Jacobian, responses, and other main quantities. These functions are the boundary between the code that is written specifically for the computation of these different quantities and the templated code, where a single code base will generate different quantities based on an Evaluation Type. That is, the different functions in this class for calculating a Residual vector or Jacobian matrix both call the same method to evaluate the PDEs, yet with a different template argument. This activates the Automatic Differentiation infrastructure in Albany (more generally called template-based generic programming). Details of this implementation are in the `PHAL_AlbanyTraits` class. The data that needs to span these two realms is packed in the `PHAL_Workset` struct.

  Also in the `src` directory are classes to handle states (fields besides the solution vector that persist between subseqent solves), "observers" that control the output and postrprocessing of solutions, and utility functions (such as a single file to manage different parallel communicator abstractions and serial code).

- `Albany/src/disc` The `disc` source code directory contains information on the global problem discretization. This includes all mesh data structures, parallel maps for so-

lution vectors, coordinates, and other fields, as well as the graph of the matrix. This information is accessed by the rest of the code through the `AbstractDiscretization` class, with accessor methods that are mainly two types: (1) `Epetra` data structures for information that will require communication between processors, and (2) standard vectors (`Teuchos::ArrayRCP` objects) for information that will not. A `Tpetra` branch of the code is under development as well.

While the global discretization abstraction insulates the rest of the code from a specific mesh database and implementation, the primary concrete implementation is through the `STK_Mesh` library from Trilinos. The STM Mesh objects can be created by reading them in from Exodus or generated by the Pamgen library, both of which use the SEACAS and IOSS tools. Alternatively, simple meshes can be generated internally in the code without the need for these libraries, including lines, rectangles (with quad or triangle elements), and brick shape meshes. These are useful for many demonstration a verification problems.

Current research efforts are developing concrete implementations for the `fmdb` mes library, for adaptivity research, and the `MPAS` mesh database for ice sheet simulations.

- `Albany/src/problems` The `problems` source code directory contains classes that inherit from an `AbstractProblem`. The (overloaded) word problem refers to a physics set, and includes things like Heat Transfer, Poisson equation, and Navier Stokes. A problem class registers a set of "evaluators" from derived from the Phalanx package in Trilinos that, in aggregate, will perform the assembly of the desired set of PDEs. All problem register the basic set of finite element evaluators as well as one or more problem-specific evaluators for computing diffusion operators, source terms, etc. Similarly, Problems define boundary condition evaluators, responses, and declare state fields (those that persist beyond one assembly).

  We do not allow the Albany user to mix-and-match all physics (PDE terms) at run time in an input file, but must construct a "problem" where the physics terms are assembled. These themselves can be written to have some degree of run time configuratbility. However, for purposes of verification, reproducibility, and limiting user error, we have made the choice to have physics sets hardwired into problem classes. The configuration of which Problems are compiled can be done with the `ENABLE_<PhysSet>` options detailed in the previous section. The Heat Transfer problem is always turned on, while the rest of the Problems in this directory can be toggled with the `ENABLE_DEMO_PDES CMake boolean`.

- `Albany/src/evaluators` The `evaluators` source code directory contains Phalanx evaluators. Phalanx is a Trilinos package that allows users to build up physics sets from individual unit operations. For instance, evaluators exist to interpolate nodal data to quadrature points, to calculate diffusion operators, or to compute a source term. The granularity of computation that occurs in an evaluator is free for a developer to choose. Common finite element operations tend to have fine granularity, where an evaluator performs a small well-defined task.

  Phalanx is written to work seamlessly with the Evaluation Type templating for full

use of automatic differetiation. The main two places where template specialization is required are the `GatherSolution` and `ScatterResidual` evaluators which seed (initialize) and extract the date to and from the automatic differentiation types. Almost all other evaluators can be written just on the generic template type. Phalanx also uses the multi-dimensional data arrays that are interoperble with those in the Trilinos `intrepid` and `shards` packages.

In this directory, there are all the evaluotors that are common to all finite element assemblies, as well as problem-specific evaluators for heat transfer, Navier-Stokes, Euler flows, Cahn Hillard problem, and other applications. The LCM, QCAD, and FELIX projects have placed evaluators specific to their applications into other directories.

- `Albany/src/responses` The `responses` source code directory contains a growing library of responses, which are post-processing routines for quantities of interest. These include several common responses that involve norms of the solution vector. Responses that required finite element information, such as integrals of quantites over the mesh, are all hooked up through the `FieldManagerScalarResponseFunction` function. This triggers an assembly phase, using evaluators, much like the Residual and Jacobian evalutations, and also make use of template-based generic programming.

- `Albany/src/LCM` The `LCM` source code directory contains application-specific code for the LCM project, as described in the introduction. It contains `problems` and `evaluators` directories that describe and implement the physics sets needed for these applications domains. The problem classes assemble PDE descriptions using evaluators both from the `src/evaluators` directory as well as the `LCM/evaluators` directory. There are additional directories for LCM development, including a `utils` directory with a Tensor library. Since LCM includes such a large code base to compile, there is a CMake configuration option `-D ENABLE_LCM:=ON` needed to enable this code base (POC: Ostien).

- `Albany/src/QCAD` The `QCAD` directory contains source code specific to the quantum device simulation and design project. It contains `problems`, `evaluators`, and `respones` subdirectors for code that only effects QCAD applciations. Currently there are several classes in the QCAD namespace that live in the Albany code base since their usage has grown beyond QCAD applications. (POC: Nielsen).

- `Albany/src/FELIX` The `FELIX` directory contains application-speciic code for the Ice Sheet dynamics application described in the introduction. It contains `problems` and `evaluators` directories that describe and implement the physics sets needed for this application domain (POC: Kalashnikova).

- `Albany/src/Hydride` The `Hydride` directory contains application-speciic code for the nuclear energy application of hydridization of cladding materials. It contains `problems` and `evaluators` directories that describe and implement the physics sets needed for this application domain (POC: Hansen).

- `Albany/src/MOR` The `MOR` directory contains source code specific to model order reduction. This includes the ability to collect and analyze snapshot information from Albany runs and produce reduced order models (POC: Cortial).

Source Code Namespaces and File Naming Conventions

There are several C++ namespaces in use in Albany. Most of the code is in the Albany namespace. This includes the general-purpose code in the `src` directory for problem setup, the processing of the mesh, and setups of many problems (physics sets). Most new development starts by default using the Albany namespace. There are opportunities to use more namespaces to clarify the modularity of the code, such as between the solver code and discretization/mesh code.

In addition to `Albany`, there are a few other namespaces in use to compartmentalize the code:

- `PHAL` This namespace (short for PHalanx-ALbany) is for the code that derives from the Phalanx base classes. This is the magic that allow for such rapid and flexible implementation of new PDEs and terms. In addition to the evaluators, also in this namespace are classes for the traits, data types, and workset that are integral to this part of the code.
- `LCM, QCAD, FELIX` These namespaces are used to visually separate code specific to those application projects. This can aid in figuring out what code is gerenal-purpose or application specific and what parts of code can be excluded from lightweight compilations.

File naming conventions are as follows.

- File Naming Conventions: prefix Most source code files adheres to the convention that the file name is the same as the class name, with the namespace as the prefix. So the class `Albany::SolverFactory` will have a filename starting with `Albany_SolverFactory`.
- File Naming Conventions: suffix The code base that is not templated uses "`.hpp`" and "`.cpp`" suffixes. For the templated code base, which is primarily the code in the `evaluators` directories, we use the following naming conventions. There is a tiny "*`file`*`.cpp`" file for explicit template instatiation, a file with "*`file`*`_Def.hpp`" extension for the definition files with the source code in it, and "*`file`*`.hpp`" for the header file.

Example/Regression Directory `Albany/examples`

`Albany/examples` This directory holds all the example problems for Albany, which also serve as the **regression tests**.

Each directory holds one or more `xml` file that is the input file for the run. Separate problems do *not* run off of separate executables. The physics set and solution method are set in the input file. The large majority of examples run of of the same `Albany` executable, which performs simulations and linearized sensititivity anaysis. Problems that perform optimization or Dakota-based UQ use the `AlbanyDakota` or `AlbanyAnalysis` executables, while intrusive UQ using the Stokhos package use the `AlbanySG` executable. Other `Main*.cpp` files can exist to create other executables, but these are currently experimental only.

The input files for each example include a `Regression Results` section which compares scalar responses, sensitivities, optimization results, and/or stochastic Galerkin results, to trusted values. Disrepencies between these results increment an integer that is the return code from `main()`. The `ctest` testing code uses this return code to decide pass/fail of tests. (We would like to extend the regression process to incorporate an `exodiff` capability to test the entire solution and the I/O capability.)

Many directories also include exodus files for specification of the finite element mesh, although some problems run from internally generated meshes. These have been partitioned using the `decomp` script from SEACAS. Both the serial and 4-processor versions of the exodus file are typically included.

Documentation Directories `Albany/doc`

- `Albany/doc` This top-level documentation directory contains useful CMake scripts with comments for configuring Trilinos, SEACAS tools, boost, and the albanyCI code. These have worked at one time on one machine, but are (unfortunately) not tested, so may drift out of date.
- `Albany/doc/webpage` The `webpage` directory contains html files for the Albany web page. (The webpage has not been filled in with much detail.) On Sandia's internal SRN network, it can be view at `https://development.sandia.gov/Albany/`. Developers are encouraged to expand the coverage and usefulness of these webpages, including the Albany code as a whole as well as the project-specific tabs.
- `Albany/doc/doxygen` The `doxygen` directory contains necessary files so that doxygen documentation of the source code will be generated. This is generated automatically and linked to from the Albany webpage.
- `Albany/doc/nightlyTestHarness` This directory contains a set of shells scripts that are used, with minor modifications, as the nightly test harness. One file with machine-specific environment variables needs to be modifies (e.g. `set_andys_env.in`) an then the test harness is run with `./run_master.sh set_andys_env.in`. This is currently run under a cron job on four platforms nightly.
- `Albany/doc/developersGuide` The directory that holds this document. Please improve and commit!

Chapter 4

Developer Workflow

This chapter consists of introductions to using Git and CMake in your development workflow. Git is the source code control tool, and CMake is for configuration management and compiling the code.

Development Using Git

The following are some workflow suggestions and general tips for using git. Of note is the autorebase feature provided by git, which in essence, upon pulling hides away your local work, updates your local repository against the remote master, and then applies your local changes "on top". This is good and safe and should be done early and often. To set this up, consult the following contents of the following gitconfig file.

```
====================== contents of ~/.gitconfig

[user]
        name = <name>
        email = <redacted>
[branch]
        autosetuprebase = always
[color]
        ui = true
        branch = auto
        diff = auto
        status = auto
[core]
        whitespace = -trailing-space,-space-before-tab
        preloadingindex = true
        preloadindex = true

[branch "master"]
        rebase = true
```

```
====================== end contents
```

It is encouraged for anyone to, at least, provide the [user] section such that the checkin messages are meaningful. You can do this by editing the .gitconfig file in your home directory (or creating it if it is not there) to include the contents above, or some subset.

The following is then a list some useful git commands. Note most of the time git and eg are interchangeable. However there are a few places where eg is arguably more useful.

- to clone Trilinos and Albany into ./Trilinos and ./Albany

  ```
  % git/eg clone <user>@software.sandia.gov:/space/git/Trilinos
  % git/eg clone <user>@software.sandia.gov:/space/git/Albany
  ```

- to pull the current repository

  ```
  % git pull
  ```

- to push changes to the master

  ```
  % git push
  ```

- to view the log of checkins to the repository

  ```
  % git/eg log
  ```

  NB: eg log is much cleaner

- to prepare local currently tracked modified files to be committed

  ```
  % git/eg add/stage <path_to_file>/<file>
  ```

  NB: git add and git stage work virtually the same in this case

- to prepare newly created files to be committed

  ```
  % git/eg add <path_to_file>/<file>
  ```

- example workflow (using eg, but git would be the same)

```
% eg pull
<edit some files>
# now build
% make
# run the tests
% ctest
# tests didn't pass so
<fix some bugs>
# build, run tests again
% make && ctest
#tests look good, commit local changes>
% eg add <path_to_file>/<file>
# check that everything is kosher
% eg status
# should say something like, "staged files ready to be committed"
% eg commit
# don't forget to write a nice one line description,
# followed by more detail if you like
# then pull and test again before you push
% eg pull
% make
% ctest
# if nothing has broken
%eg push
```

- hypothetically, pulled changes don't compile (this will happen)

```
# how to recover
# check the log to see if there is an obvious checkin causing the error
# there will be a tag, something like master~#
# then use the following syntax, I'll use #=5 for demonstration
# here eg is required
% eg reset --working-copy master~5
```

- another hypothetical, you have lots of changes locally, but you'd like to pull, and you haven't committed anything

```
# you can use the stash command
% eg stash save workInProgress
# now you can pull without issue
% eg pull
# then if you want to, you can apply your changes
% eg stash apply workInProgress
# then resolve conflicts as necessary
```

Development Using CMake

Need to add how to have a conditional ENABLE_MYCRUD compile option.

The following are some general steps to add new evaluators or problems, as well as new test problems to Albany CMakeLists. As an example, there are two newly created source and header files belonging to Albany LCM `myNewProblem.cpp, myNewProblem.hpp` that you would like to add to the CMakeLists (same steps apply for other types of Albany problems or evaluators).

- Add new problem files in `<path_to_Albany_directory>/src/CMakeList.txt` at the correspoinding section (in this example `ALBANY_LCM`)

```
# LCM
IF(ALBANY_LCM)
 SET(LCM_DIR ${Albany_SOURCE_DIR}/src/LCM)
 # LCM problems
 SET(SOURCES ${SOURCES}
 ${LCM_DIR}/problems/myNewProblem.cpp
 )
 SET(HEADERS ${HEADERS}
 ${LCM_DIR}/problems/myNewProblem.hpp
 )
ENDIF()
```

- To add a test problem to Albany examples, first create a new directory `<path_to_Albany_directory>`

- Within the new directory, create necessary input files to run the example, as well as a `CMakeLists.txt` file. A simple example `CMakeLists.txt` file may include the following

```
# 1. Copy Input file from source to binary dir
configure_file(${CMAKE_CURRENT_SOURCE_DIR}/input.xml
               ${CMAKE_CURRENT_BINARY_DIR}/input.xml COPYONLY)
# 2. Name the test with the directory name
get_filename_component(testName ${CMAKE_CURRENT_SOURCE_DIR} NAME)
# 3. Create the test with this name and standard executable
add_test(${testName} ${Albany.exe})
```

  NB: you may also copy and paste existing test examples in Albany, and modify according to your new problem.

- Add in `<path_to_Albany_directory>/examples/CMakeLists.txt` the directory name of the newly created example at corresponding sectioin (again, in this example `ALBANY_LCM`).

```
IF(ALBANY_LCM)
 add_subdirectory{myNewProblem}
ENDIF()
```

- Compile and test to make sure the newly added problem passes ctest.

  ```
  % cd <path_to_Albany_directory>/build
  % make && ctest
  ```

  Once it passes ctest, you can push your changes to the master repository using the steps listed in the previous section.

Chapter 5

Albany Mailing Lists

All those interested in keeping up with Albany development should subscribe to the following mailman lists:

- albany-checkins `https://software.sandia.gov/mailman/listinfo/albany-checkins`
- albany-developers `https://software.sandia.gov/mailman/listinfo/albany-developers`

Developers making commits with any frequency should subscribe to the nightly test results (about 3 emails per night) since these test build/configuration/platform options that] might not be part of your workflow.

- albany-regression `https://software.sandia.gov/mailman/listinfo/albany-regression`

Chapter 6

Albany Copyright and Licensing Info

`Albany 2.0` has been approved for open source release with a Publically Available designation. There is a file `license.txt` with the full text. All other source code has a short banner that should be added to the top of any new source code files. We will re-assert copyright for major releases, but not minor releases.

The license is (as one should expect) copied from what is used in a typical Trilinos package, which is a BSD-style license. Since part of the license states that the details of the license will be included in documentation, we attach it here.

```
**********************************************************************
     Albany 3.0:  Copyright 2016 Sandia Corporation

Under the terms of Contract DE-AC04-94AL85000 with Sandia Corporation,
the U.S. Government retains certain rights in this software.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the Corporation nor the names of the
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY SANDIA CORPORATION "AS IS" AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SANDIA CORPORATION OR THE
```

```
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Questions? Contact Andy Salinger (agsalin@sandia.gov)
*************************************************************************
```

DISTRIBUTION:

	MS	,
		,
1	MS 0899	Technical Library, 8944 (electronic copy)


Sandia National Laboratories