



Summary and Final Comments

Michael A. Heroux

How to Create a Trilinos (Compatible) Package

- Two primary cases
 - ◆ Using Autotools with an existing package
 - ◆ Starting a new package using Autotools
 - ◆ Both cases are similar

- In either case, the package might be
 - ◆ Stand alone
 - ◆ Used via Trilinos/packages/external
 - ◆ Added to Trilinos

How to Create a Trilinos (Compatible) Package

- Look at the new_package package
- Customize the following files for your package
 - ♦ configure.ac
 - ♦ Makefile.am
 - ♦ src/Makefile.am
 - ♦ test/Makefile.am
 - ♦ example/Makefile.am
 - ♦ Makefile.export.<package>.in
 - ♦ (Some of the necessary changes can be made using scripts supplied by new_package)
- Additional instructions are supplied with new_package

Adding Files to the Build System and Tarball

- Library source files:

```
CORE = \  
    $(srcdir)/Epetra_BLAS.cpp \  
    ...  
    $(srcdir)/Epetra_Object.cpp
```

```
CORE_H = \  
    $(srcdir)/Epetra_BLAS.h \  
    ...  
    $(srcdir)/Epetra_ConfigDefs.h
```

- ♦ Conditionally compiled files listed with 'EXTRA_' prefix
- ♦ Don't forget to list header files!

Adding Files to the Build System and Tarball

- Makefile.am/.in:
 - ◆ Add the directory the new files are in to 'SUBDIRS' in the Makefile.am one level up
 - SUBDIRS = DIR1 DIR2
 - ◆ Add the Makefile that will be generated to 'AC_CONFIG_FILES' in configure.ac
 - AC_CONFIG_FILES([Makefile ... src/Makefile ...])
 - ◆ Don't forget to 'cvs add' both files
 - ◆ ./bootstrap

- Other types of files (scripts, plain text, etc):
 - ◆ Add the name of the file to EXTRA_DIST
 - EXTRA_DIST = script1 README ...
 - ◆ ./bootstrap

Adding Configure Options

- `TAC_ARG_ENABLE_CAN_USE_PACKAGE(epetra, Teuchos, ...)`
 - ◆ ‘`#ifdef HAVE_EPETRA_TEUCHOS`’ in source code
- `TAC_ARG_ENABLE_FEATURE_SUB(epetra, abc, ...)`
 - ◆ ‘`#ifdef HAVE_EPETRA_ARRAY_BOUNDS_CHECK`’ in source

Adding Configure Options

- `TAC_ARG_WITH_PACKAGE(zoltan, [Enable Zoltan interface support], ZOLTAN, no)`
 - ◆ `AM_CONDITIONAL(HAVE_ZOLTAN, [test "X$ac_cv_use_zoltan" != "Xno"])`
 - 'if HAVE_ZOLTAN' in Makefile.am
- `AC_SEARCH_LIBS(pow,[m],,AC_MSG_ERROR(Cannot find math library))`

Using Makefile.export for Tests / Examples

- Makefile.am:

```
include $(top_builddir)/Makefile.export.epetra
```

```
EXEEXT = .exe
```

```
noinst_PROGRAMS = CrsMatrix_test
```

```
CrsMatrix_test_SOURCES = $(srcdir)/cxx_main.cpp
```

```
CrsMatrix_test_DEPENDENCIES=$(top_builddir)/src/libepetra.a
```

```
CrsMatrix_test_CXXFLAGS = $(EPETRA_INCLUDES)
```

```
CrsMatrix_test_LDADD = $(EPETRA_LIBS)
```



Benefits of Generic Programming

- 1) Generic programming techniques ease the implementation of complex algorithms.
- 2) Developing algorithms with generic programming techniques is easier on the developer, while still allowing them to build a flexible and powerful software package.
- 3) Generic programming techniques also allow the user to leverage their existing software investment.

Caveat: It's not as easy as taking a piece of code and adding:

```
template <typename OT, typename ST>
```

More work has to be done to handle “numeric traits”



Teuchos Numeric Traits

- OrdinalTraits
 - ◆ zero & one
 - ◆ int & long int
- ScalarTraits
 - ◆ zero, one, magnitude type, absolute value, conjugate, square root, random number generator, ...
 - ◆ `std::numeric_limits`
 - ◆ `float`, `double`, `complex<float>`, and `complex<double>`
- Arbitrary precision arithmetic (ARPREC, GMP)
- Templated BLAS/LAPACK wrappers, serial dense matrix/vector

Generic programming technique that uses templated interfaces to define the standard behavior of datatypes.



Teuchos Templated BLAS Example

```
double DNRM2( const int n, const double* x, const int incx) const
{
    int i, ix = 0;
    double result 0.0;
    if ( n > 0 )
    {
        // Set the initial index.
        if (incx < 0) { ix = (-n+1)*incx; }

        for(i = 0; i < n; ++i)
        {
            result += x[ix] * x[ix];
            ix += incx;
        }
        result = std::sqrt(result);
    }
    return result;
} /* end NRM2 */
```



Teuchos Templated BLAS Example

```
typedef ScalarTraits<ScalarType>::magnitudeType MagnitudeType
template<typename OrdinalType, typename ScalarType>
MagnitudeType BLAS<OrdinalType, ScalarType>::NRM2( const OrdinalType n,
                                                    const ScalarType* x,
                                                    const OrdinalType incx) const
{
  OrdinalType izero = OrdinalTraits<OrdinalType>::zero();
  OrdinalType ione = OrdinalTraits<OrdinalType>::one();
  MagnitudeType result = ScalarTraits<MagnitudeType>::zero();
  OrdinalType i, ix = izero;
  if ( n > izero )
  {
    // Set the initial index.
    if (incx < izero) { ix = (-n+ione)*incx; }

    for(i = izero; i < n; i++)
    {
      result += ScalarTraits<ScalarType>::magnitude(ScalarTraits<ScalarType>::conjugate(x[ix]) * x[ix]);
      ix += incx;
    }
    result = ScalarTraits<MagnitudeType>::squareroot(result);
  }
  return result;
} /* end NRM2 */
```

Themes for FY08/09

- Redefinition of Trilinos scope beyond solvers.
- Next steps in packaging and distribution.
- Continued outreach to other communities
- Rethinking source management.
- Formalizing App-Trilinos relationship.
- Post-delivery maintenance improvements.
- Trilinos Advisory Group.
- More (will emerge during this meeting).

Scope of Trilinos

- Addition of Sacado, Zoltan, FEI, Intrepid, phdMesh: Not solvers.
- Framework support natural.
- Rephrasing of project goals, descriptions underway.
- Grouping of packages into meta-packages: At least conceptually.

A Tale of Two Trilinoses

(with apologies to Dickens)

- Trilinos is both:
 1. Software Engineering Environment
 2. Delivery mechanism for compatible tools.
- Expansion of scope clearly reasonable regarding item 1.
- Item 2 requires rethinking of packaging, release processes.

Packaging and Distribution

- Mac and Windows are ever more popular development environments.
- Goal: Provide click-install capabilities for Mac OS, MS Visual Studio, Linux COE.

Outreach

- Trilinos packages part of SciDAC:
 - ◆ ITAPS, CSCAPES, TOPS-2.
 - ◆ Opportunity to serve broader DOE community.
- Trilinos popular in universities:
 - ◆ Single largest sector of users.
- Trilinos part of several industrial efforts.
 - ◆ Improves capabilities.
 - ◆ Amortizes costs over broader funding sources.
- Elevates certain activities:
 - ◆ Fortran accessibility.
 - ◆ Packaging & distribution.

Source Management

- Think of repository as a database.
- Logical collections gathered dynamically.
- Consider use of multiple source management tools:
 - ◆ Local vs. global management.
 - ◆ Fully distributed.
- Certainly svn is option, but looking at all options.

App-Trilinos Relationship

- How should apps to interact with Trilinos?
- Many facets:
 - ◆ Interfaces: concrete vs. abstract.
 - ◆ Testing: How frequently, who does it.
 - ◆ Distribution: Trilinos separate or integrated.
 - ◆ ...
- Goal: Any two applications should differ only in:
 - ◆ Essential needs due to formulation.
 - ◆ Specific uses of enabling technologies.

Conclusions

- Trilinos: Large and growing collection.
- Can be daunting to first-time user.
 - ◆ C++, OO, lots of names.
 - ◆ But payoff is huge after initial investment.
 - ◆ Alternative: Learn about same capabilities in 45 separately developed projects.
- Trilinos provides many unique capabilities.
 - ◆ Enabling technology libraries for most basic application needs.
 - ◆ Interoperable, not interdependent.
 - ◆ Truly extensible.
 - ◆ Adapting to modern architectures.
 - ◆ Adapting to meet needs of DOE Science community: PETSc, Fortran.