USING BILDER TO BUILD TRILINGUALLY

T. Austin, S. Kruger, R. Pundaleeka
Goal: Get You to Using Trilinos Today

- Some people can be overwhelmed with complexity of build systems for scientific software.

- Building third-party libraries (MPI, lapack, SuperLU) and getting the dependencies correct can be a nightmare (especially true for Windows).

- Goal is to help you avoid nightmare step by making the build and install process for Trilinos as easy as possible.

- Let Bilder do all of the work for you so you can solve real problems rather than getting tangled up in frustrating compiler and library issues.
What is Bilder?
A meta-build system for scientific software

- It deals with code *packages*, not code source.
- It is geared to building *chains* of dependencies of arbitrary length.
- It is cross-platform with no compiler assumptions.
- It is hosted at sourceforge ([http://sourceforge.net/p/bilder](http://sourceforge.net/p/bilder)).
- It is for scientific software
  - Fortran is special
  - MPI is special
  - Handle the diamond-structure dependencies commonly found in scientific software
    - IO libraries (netcdf, HDF5) and math libraries (blas,lapack) are common dependencies
- It is *not* a package manager system like Gentoo’s portage or MacPorts.
Originally developed to solve problems with FACETS: code-coupling framework in the fusion community

FACETS
UEdge  GYRO  NUBEAM  ...
PETSc  MUMPS
BLAS/LAPACK  NetCDF
HDF5
MPI (openmpi)
Compiler

Legacy fusion codes:
generally crappy, but contains lots of knowledge that we want to save

Having individual build systems find different HDF5 libraries is very bad!

FACETS really has ~30 packages in it’s build chain

May or may not be part of build chain
What are the common features in building a package?

**Bilder**: Controls the step of building and installing individual packages

- **Fetch**: Tarball or use repo?
  - Tarballs come from “numpkgs” repo at Tech-X
- **Preconfig**: Do we need to patch for a special system?
- **Configure**: Install tarballs in one location and repos in another?
- **Build**: Do we have to do something special?
- **Test**: Is the build working properly?
- **Install**: Anything to do afterwards, like fix permissions?
Using Bilder to build Trilinos
Step 1: Setup

- Make sure you have your target machine ready:
  
  http://sourceforge.net/p/bilder/wiki/Preparing%20your%20machine%20for%20Bilder/

- Obtain an account on github (open to anyone):

- Make the following calls from the command line (bash shell):

  % git clone https://USERNAME@github.com/Tech-XCorp/trilinosall.git trilinosall
  % cd trilinosall
  % ./externalrepos.sh
  # Obtain a recent version (11.0.3) of Trilinos or get the repo from publicTrilinos
Using Bilder to build Trilinos
Step 1: Setup

Note: rst2html.py README.rst > README.html generates HTML instructions as well. See file:///Users/austin/Projects/Trilinos/trilinosall/README.html.
## Packages relevant to Trilinos

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDF5</td>
<td>1.8.7-9</td>
<td>Yes</td>
</tr>
<tr>
<td>Qt</td>
<td>4.8.1</td>
<td>Yes</td>
</tr>
<tr>
<td>Thrust</td>
<td>1.6.0</td>
<td>Yes</td>
</tr>
<tr>
<td>Zlib</td>
<td>1.2.6</td>
<td>Yes</td>
</tr>
<tr>
<td>PETSc</td>
<td>3.2 or 3.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Dakota</td>
<td>5.2</td>
<td>?</td>
</tr>
<tr>
<td>Boost/Boostlib</td>
<td>1_47_0 (1_50_0)</td>
<td>Yes</td>
</tr>
<tr>
<td>netcdf</td>
<td>4.1.12</td>
<td>Yes</td>
</tr>
<tr>
<td>SuperLU</td>
<td>4.1</td>
<td>Yes</td>
</tr>
<tr>
<td>SuperLUDist</td>
<td>3.2</td>
<td>Yes</td>
</tr>
<tr>
<td>METIS</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>HYPRE</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

To see all packages supported: `ls bilder/packages`
Packages relevant to Trilinos
#!/bin/bash

SUPERLU_BLDRVERSION=${SUPERLU_BLDRVERSION:-"4.1"}

SUPERLU_BUILDS=ser,sersh
SUPERLU_DEPS=cmake,atlas,lapack,clapack_cmake
SUPERLU_UMASK=002

buildSuperlu() {
  if bilderUnpack superlu; then
    if BilderConfig -c superlu ser; then
      bilderBuild superlu ser
    fi
    if BilderConfig superlu sersh "-DBUILD_SHARED_LIBS:BOOL=ON"; then
      bilderBuild superlu sersh
    fi
  fi
}

The two main scripts are:

- `mktriall.sh`
  Main bilder script that fine-tunes many of the build aspects.

- `mktriall-default.sh`
  Bilder script for handling default parameters for simplifying the builds, including the default locations at LCFs.

- For both scripts, `--h` or `--help` commands will show options.

- To build trilinos with all the default builds and third party dependencies, first *print* what the default will do:
  ```bash
  ./mktriall-default.sh -p
  ```

source /Users/austin/Projects/Trilinos/trilinosall/bilder/runnr/runnrfcns.sh
Command is
./mktriall.sh -k /Users/austin/software -i /Users/austin/software --e austin@txcorp.com
runBilderCmd exiting with 0.
Understanding Bilder output:
Terminology

- **PROJECT_DIR**
  This is the directory location of this file.

- **INSTALL_DIR**
  This is where trilinos will be installed (./mktrilions.sh –i INSTALL_DIR)

- **CONTRIB_DIR**
  This is where TPLs from tarballs will be installed (-k CONTRIBUTION_DIR)
  This may equal the INSTALL_DIR

- **BUILD_DIR**
  This is where the builds are location (-b BUILD_DIR)
  Typically ``$PROJECT_DIR/builds``

For example, we have by default trilinosall/builds where we would see SuperLU and SuperLU_Dist builds.

Typically use ~/Software as INSTALL_DIR and CONTRIBUTION_DIR.
Understanding Bilder output:
Key files

• Key output files:
  
  `$BUILD_DIR/mktriall.log`
  `$BUILD_DIR/mktriall-summary.txt`
  `$BUILD_DIR/trilinos-chain.txt`

• For each package (e.g., trilinos)

  `$BUILD_DIR/trilinos/<build>/<hostname>-<pkg>-<build>-<step>.txt`
  
  • E.g., `$BUILD_DIR/trilinos/ser/iter.txcorp.com-trilinos-ser-build.txt`

  • To debug, it is help to use the scripts that generated the build:

  What is wrong?` cd $BUILD_DIR/trilinos/ser`
  Can I fix?` cat iter.txcorp.com-trilinos-ser-build.txt`
  Did it work?` vi iter.txcorp.com-trilinos-ser-build.sh`
  ` iter.txcorp.com-trilinos-ser-build.sh`
Customizing Trilinos builds

To set up necessary builds and third party dependencies, create a configuration file called ``trilinos.conf`` in $PROJECT_DIR

- cp trilinos.conf.example trilinos.conf

Key variables:

- TRILINOS_BUILDS
  Which types of builds do. Possible choices are ser, par, sersh, parsh where the sh suffice refers to shared builds

- TRILINOS_DEPS
  To turn on and off TPL dependencies. Needs to be coordinated with TRILINOS_ADDL_ALLARGS potentially

- TRILINOS_ADDL_ALLARGS
  Arguments used by all builds. Generally used to turn on and off trilinos packages and TPL.

- TRILINOS_<BUILD>_OTHER_ARGS
  Arguments for the individual builds.
Sample trilinos.conf

TRILINOS_BUILDS="ser,par"
TRILINOS_DEPS="swig,openmpi,boost,hdf5"
TRILINOS_ADDL_SHARGS="-DTrilinos_ENABLE_Amesos:BOOL=ON"
Building other packages

• Bilder has other packages that you may want to build.
• mktriall.sh can take as an argument a different package
• For example, ipython has a pretty long build chain that includes almost all useful scientific python packages
  mktriall-default.sh -n - ipython
will build the ipython build chain in the default locations
Conclusions and further work

- Bilder is a useful tool for building dependency chains on different platforms.
- We have “bilderized” trilinos to make it easier for people to build the trilinos build chain.
- Customizing your build to choose your dependencies is possible with the trilinos.conf file.
- Bilder documents all the steps thoroughly to allow debugging of any problems that arise.
  - Any problems can be sent to developer@txcorp.com
- We welcome feedback and suggestions for improvements.