Best Practices for HPC Software Developers Webinar Series

Session 4: Testing and Documenting Your Code

We will also give a half day tutorial on testing at SC16: “Testing of HPC Scientific Software”

Welcome! We will begin soon

- We want this webinar to be interactive, and we encourage questions
  - But we need to keep everyone’s mic muted (too many participants)
  - Please use the Zoom Q&A tool to submit questions
  - Or use type them into this Google Doc: http://bit.ly/hpcbp-qa
  - Use the Zoom Chat tool for other issues

- Slides and a recording will be available from the OLCF training web site: https://www.olcf.ornl.gov/training-event/webinar-series-best-practices-for-hpc-software-developers

- We want to improve this series. Please send feedback to HPCBestPractices+session04@gmail.com

2016-06-16
Testing and Documenting your Code

Alicia Klinvex

Sandia National Labs

June 15, 2016
Outline

- Testing
  - Why testing is important
  - Types of tests
  - Testing tips
  - How Trilinos is tested
  - Code coverage

- Documentation
  - Why documentation is important
  - Types of documentation
  - How Trilinos is documented
  - Documentation generators
TESTING
Why testing is important: the protein structures of Geoffrey Chang

- Some inherited code flipped two columns of data, inverting an electron-density map
- Resulted in an incorrect protein structure
- Resulted in 5 retracted publications
  - One was cited 364 times
- Many papers and grant applications conflicting with his results were rejected
Why testing is important: the 40 second flight of the Ariane 5

- Ariane 5: a European orbital launch vehicle meant to lift 20 tons into low Earth orbit
- Initial rocket went off course, started to disintegrate, then self-destructed less than a minute after launch
- Seven variables were at risk of leading to an Operand Error (due to conversion of floating point to integer)
  - Four were protected
- Investigation concluded insufficient test coverage as one of the causes for this accident
- Resulted in a loss of $370,000,000.
Why testing is important: the Therac-25 accidents

- Therac-25: a computer-controlled radiation therapy machine
- Minimal software testing
- Race condition in the code went undetected
- Unlucky patients were struck with approximately 100 times the intended dose of radiation, ~15,000 rads
- Error code indicated that no dose of radiation was given, so operator instructed machine to proceed
  - Documentation gave no indication that the frequent malfunctions of the machine could place a patient at risk
  - See also: why documentation is important
- Recalled after six accidents resulting in death and serious injuries
Granularity of tests

- **Unit tests**
  - Test individual functions or classes
  - Build and run fast
  - Localize errors

- **Integration tests**
  - Test interaction of larger pieces of software

- **System-level tests**
  - Test the full software system at the user interaction level
Types of tests

- Verification tests
  - Does the code implement the intended algorithm correctly?
  - Check for specific mathematical properties

- Acceptance tests
  - Assert acceptable functioning for a specific customer
  - Generally at the system-level

- Regression (no-change) tests
  - Compare current observable output to a gold standard
  - Must independently verify that the gold standard is correct

- Performance tests
  - Focus on the runtime and resource utilization
  - Nothing to do with correctness

- Installation tests
  - Verify that the configure-make-install is working as expected
CSE testing challenges

- Floating point issues
  - Different results
    - On different platforms
    - On different runs (due to multi-processor computation)
  - Ill-conditioning can magnify these small differences
    - Final solution may be different
    - Number of iterations may be different
  - Performing a diff is bad
- Non-unique solutions
CSE testing challenges

- Scalability testing
  - Difficult to get accurate data on a shared machine
  - Getting access to many processors on a parallel machine is expensive
    - Many supercomputing facilities discourage routine scalability testing
    - Large jobs may sit in the queue for quite some time
  - How do you scale a problem for weak scaling studies?
    - A more refined problem may not have the same condition number
Testing tips

- Ideal time to build a test suite is during development
  - Ensures that new code does not break existing functionality
- Failing tests should help you identify what part of the code needs to be fixed
- Software should be tested regularly
- Develop a consistent policy on dealing with failed tests
  - Use an issue tracking system
  - Add a regression test after the issue is fixed
- Run a regression test suite when checking in new code
- Avoid zero-diffing tests against gold standard output
  - spiff ([https://github.com/dontcallmedom/spiff](https://github.com/dontcallmedom/spiff))
What is Trilinos?

- A collection of libraries intended to be used as building blocks for the development of scientific applications
- Organized into 66 packages
  - Linear solvers
  - Nonlinear solvers
  - Eigensolvers
  - And more!
- 10,000+ commits
- 135 contributors (according to github)
- Millions of lines of code
How is Trilinos tested?

- Trilinos has 1500 tests between its 66 packages
- Developers are strongly advised to run a checkin test script when committing
  - Detects which packages were modified by your commits
  - Determines which packages you potentially broke
  - Configures, builds, and tests those packages
    - On success, pushes to repo
    - On failure, reports why it failed
  - Useful for ensuring your changes don’t break another package
  - May take a while, but many people run it overnight
- Automated testing on a variety of different platforms
Why do we do automated testing if everyone uses the checkin script?

- May test a different set of packages
- May test different environments
  - Do your changes work with Intel compilers as well as GNU?
  - Do your changes work on a mac?
  - Do your changes work with CUDA?
- Identifies a small set of commits that could have broken a build or test
  - Average 12 commits per day
  - Identifies the person who knows how to un-break it
- Bugs are easier to fix if caught early
Checkin test script examples

- Example 1: a harmless change to a comment
- Example 2: breaking the build
- Example 3: breaking some tests
Example 1: a harmless change
Example 1: a harmless change

```bash
$ amklinv@9995092 CHECKIN$ ./checkin-test-amklinv --do-all

Script: checkin-test.py
--src-dir="/home/amklinv/TrilinosDir/github/Trilinos"
--default-builds=MPI_DEBUG,SERIAL_RELEASE
--extra-repos-file=
--extra-repos-type=
--extra-repos=
--require-extra-repos-exist
--enable-packages=
--disable-packages='FEI,Moertel,STK,Phalanx,PyTrilinos'
--enable-all-packages="auto"
--enable-fwd-packages
--continue-if-no-updates
--continue-if-no-changes-to-push
--continue-if-no-enables
--extra-cmake-options=""
--test-categories='BASIC'
--make-options='--j40'
--ctest-options='-E "(Anasazi_Epetra_BlockDavidson_test|Anasazi_Epetra_BlockDavidson_auxtest_MPI_4|Anasazi_Epetra_GeneralizedDavidson_solver_test_MPI_4|Anasazi_BlockDavidsonThyra_test_MPI_4|NOX_Thyra_Heq_MPI_1|Rythmos_BackwardEuler_ConvergenceTest_MPI_1|Rythmos_IntegratorBuilder_ConvergenceTest_MPI_1|NOX_Thyra_Heq)"
--ctest-timeout=60.0
--no-show-all-tests
--st-extra-builds=''
--extra-builds=''
--send-email-to='amklinv@sandia.gov'
--skip-case-send-email
--send-build-case-email=always
--send-email-for-all
--send-email-to-on-push='trilinos-checkin-tests@software.sandia.gov'
--no-force-push
--do-push-readiness-check
--rebase
--append-test-results
--do-all

Starting time: Thu Apr 21 16:11:38 MDT 2016
```
Example 1: a harmless change

Note that the checkin script correctly identified what was modified.
Example 1: a harmless change

Configure, build, and test passed for MPI_DEBUG

E) Analyze the overall results and send email notification (MPI_DEBUG) ...

E.1) Determine what passed and failed ...

The pull passed!

The configure passed!

The build passed!

`testResultsLine = '100% tests passed, 0 tests failed out of 237'

All of the tests ran passed!

E.2) Construct the email message ...

`subjectLine = 'passed: Trilinos/MPI_DEBUG: passed=237,notpassed=0'

Running: touch email.success

E.3) Send the email message ...

Running: mailx -s "passed: Trilinos/MPI_DEBUG: passed=237,notpassed=0" amklinv@sandia.gov < email.out

***

*** Doing build and test of SERIAL_RELEASE ...

***
Example 1: a harmless change

We are ready to push because all tests passed
Example 2: broken build

Missing semicolon at the end of the class. This will break the build.
Example 2: broken build

The checkin script detected that I broke the build
Example 2: broken build

Checkin script also creates a log file with the error
Example 3: broken tests

Added a logic error to the code.
Example 3: broken tests

The checkin script detected that I broke several tests.
Example 3: broken tests

The log file tells us which tests were broken
# Trilinos automated testing

## Project Overview

<table>
<thead>
<tr>
<th>Project</th>
<th>Configure Error</th>
<th>Configure Warning</th>
<th>Configure Pass</th>
<th>Build Error</th>
<th>Build Warning</th>
<th>Build Pass</th>
<th>Test Not Run</th>
<th>Test Fail</th>
<th>Test Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trilinos</td>
<td>1</td>
<td>531</td>
<td>530</td>
<td>0</td>
<td>272</td>
<td>257</td>
<td>0</td>
<td>14</td>
<td>3976</td>
</tr>
</tbody>
</table>

## SubProjects Overview

<table>
<thead>
<tr>
<th>Project</th>
<th>Configure Error</th>
<th>Configure Warning</th>
<th>Configure Pass</th>
<th>Build Error</th>
<th>Build Warning</th>
<th>Build Pass</th>
<th>Test Not Run</th>
<th>Test Fail</th>
<th>Test Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teuchos</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>227</td>
</tr>
<tr>
<td>ThreadPool</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>564</td>
</tr>
<tr>
<td>Sacado</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>RTOp</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>564</td>
</tr>
<tr>
<td>Kokkos</td>
<td>0</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>Epetra</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>244</td>
</tr>
<tr>
<td>Zoltan</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>13</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>135</td>
</tr>
<tr>
<td>Shards</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GlobiPack</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Trilinos automated testing

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Update</th>
<th>Configure</th>
<th>Build</th>
<th>Test</th>
<th>Build Time</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>artemis.srn.sandia.gov</td>
<td>Linux-intel-15.0.2-MPI_RELEASE_DEV_DownStream_ETI_SERIAL-OFF_OPENMP-ON_PTHREAD-OFF_CUDA-OFF_COMPLEX-OFF</td>
<td>68</td>
<td>1</td>
<td>140</td>
<td>0</td>
<td>3</td>
<td>6 hours ago (44 labels)</td>
</tr>
<tr>
<td>lightsaber.srn.sandia.gov</td>
<td>Linux-GCC-4.7.2-RELEASE_DEV_MueLu_Matlab</td>
<td>69</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>10 hours ago (25 labels)</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_Barker</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>0</td>
<td>9 hours ago (25 labels)</td>
</tr>
<tr>
<td>hansel.sandia.gov</td>
<td>Linux-GCC-4.4.7-MPI_OPT_DEV_XYCE</td>
<td>121</td>
<td>0</td>
<td>70</td>
<td>0</td>
<td>0</td>
<td>9 hours ago (13 labels)</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_KLU2</td>
<td>69</td>
<td>0</td>
<td>225</td>
<td>0</td>
<td>0</td>
<td>8 hours ago (25 labels)</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_ExtraTypes_El</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>0</td>
<td>8 hours ago (25 labels)</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-SERIAL_DEBUG_DEV_MueLu_ExtraTypes</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>0</td>
<td>7 hours ago (25 labels)</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-SERIAL_RELEASE_DEV_MueLu_Experimental</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>0</td>
<td>6 hours ago (25 labels)</td>
</tr>
</tbody>
</table>
Trilinos automated testing

- Several Amesos2 (direct solver) tests are broken.
  - Are any of its dependencies broken?
    - Yes, there is a broken Epetra (basic linear algebra) test
    - Maybe this broke Amesos2
Trilinos automated testing

- Which tests were broken in Amesos2?

Testing started on 2016-06-06 07:42:35

Site Name: enigma.sandia.gov
Build Name: Linux-GCC-4.8.3-SERIAL_DEBUG_DEV_MueLu_ExtraTypes
Total time: 16s 840ms
OS Name: Linux
OS Platform: x86_64
OS Release: 3.10.0-229.4.2.el7.x86_64
OS Version: #1 SMP Fri Apr 24 15:26:38 EDT 2015
Compiler Version: unknown

3 tests failed.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Time</th>
<th>Details</th>
<th>Labels</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amesos2_Epetra_RowMatrix_Adapter_UnitTests_MPI4</td>
<td>Failed</td>
<td>1s 860ms</td>
<td>Completed (Failed)</td>
<td>Amesos2</td>
<td>Broken</td>
</tr>
<tr>
<td>Amesos2_Epetra_MultiVector_Adapter_UnitTests_MPI4</td>
<td>Failed</td>
<td>1s 980ms</td>
<td>Completed (Failed)</td>
<td>Amesos2</td>
<td>Broken</td>
</tr>
<tr>
<td>Amesos2_Tpetra_CrsMatrix_Adapter_UnitTests_MPI4</td>
<td>Failed</td>
<td>1s 900ms</td>
<td>Completed (Failed)</td>
<td>Amesos2</td>
<td>Broken</td>
</tr>
</tbody>
</table>
Trilinos automated testing

- If you may have broken something, you will get an email about it

CDash <trilinos-regression@sandia.gov> 4:05 AM (5 hours ago) 🌟
to anasazi-regres.

A submission to CDash for the project Trilinos has failing tests.
You have been identified as one of the authors who have checked in changes that are part of this submission or you are listed in the default contact list.


Project: Trilinos
SubProject: Anasazi
Site: [artemis.srn.sandia.gov](http://www.artemis.srn.sandia.gov)
Build Name: Linux-intel-15.0.2-MPI_RELEASE_DEV_DownStream_ETI_SERIAL-OFF_OPENMP-ON_PTHREAD-OFF_CUDA-OFF_COMPLEX-OFF
Build Time: 2016-06-06T03:59:42 MDT
Type: Nightly
Tests failing: 1

*Tests failing*
How do you motivate somebody to write all those tests?

- Tests protect YOU from other people from breaking your work
  - If someone else’s changes break your code, they are responsible for fixing it
- You may already have some
  - Drivers for generating conference or paper results
    - Just reduce the problem size
  - User submitted bugs
    - Ask for a file that reproduces the issue
    - These make great regression tests
  - Examples
    - Add a pass/fail condition and you have a test
How do I determine what other tests I need?

- **Code coverage tools**
  - Expose parts of the code that aren’t being tested
  - **gcov**
    - standard utility with the GNU compiler collection suite
    - counts the number of times each statement is executed
  - **lcov**
    - a graphical front-end for gcov

Similar tools exist for mac and windows
How to use lcov

- Compile and link your code with --coverage flag
  - It’s a good idea to disable optimization
- Run your test suite
- Collect coverage data using lcov
- Generate html output using genhtml
A simple example

```cpp
#include<iostream>
#include "isEven.hpp"

int main()
{
    int num = 8;

    if(isEven(num))
        std::cout << num << " is an even number.\nTEST PASSED";
    else
        std::cout << num << " is an odd number.\nTEST FAILED";

    return 0;
}

bool isEven(int x) { 
    if(x%2 == 0)
        return true;
    return false;
}
```
A simple example

- Compile and link with --coverage flag
  - `g++ --coverage evenExample.cpp -o evenExample`
  - This creates a file called evenExample.gcno
- Run the test
  - `./evenExample`
  - This creates a file called evenExample.gcda
- Collect coverage data using lcov
  - `lcov --capture --directory . --output-file evenExample.info`
  - This creates evenExample.info
- Generate html output using genhtml
  - `genhtml evenExample.info --output-directory evenHTML`
  - This generates html files in the directory evenHTML
A simple example

LCOV - code coverage report

Current view: **top level** - /home/amklinv/IDEAS/testingTalk/examples/simpleExample

Test: **evenExample.info**
Date: **2016-05-24 14:13:07**

<table>
<thead>
<tr>
<th>Filename</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>evenExample.cpp</td>
<td>85.7 %</td>
<td>100.0 %</td>
</tr>
<tr>
<td>isEven.hpp</td>
<td>75.0 %</td>
<td>100.0 %</td>
</tr>
</tbody>
</table>

Generated by: **LCOV version 1.12-4-g04a3c0e**

This is the file we’re testing
A simple example

We never tested this line of code (which activates when x is odd)
Let’s add another test

```cpp
#include<iostream>
#include "isEven.hpp"

int main()
{
    int num = 7;

    if(isEven(num))
        std::cout << num << " is an even number.\nTEST FAILED";
    else
        std::cout << num << " is an odd number.\nTEST PASSED";

    return 0;
}
```

```cpp
bool isEven(int x)
{
    if(x%2 == 0)
        return true;
    return false;
}
```
A simple example

- Compile and link with --coverage flag
  - `g++ --coverage oddExample.cpp -o oddExample`
  - This creates a file called `oddExample.gcno`
- Run the test
  - `./oddExample`
  - This creates a file called `oddExample.gcda`
- Collect coverage data for BOTH TESTS using lcov
  - `lcov --capture --directory . --output-file twoExamples.info`
  - This creates `twoExamples.info`
- Generate html output using genhtml
  - `genhtml twoExamples.info --output-directory totalHTML`
  - This generates html files in the directory `totalHTML`
A simple example

### LCOV - code coverage report

Current view: top level - /home/amklinv/IDEAS/testingTalk/examples/simpleExample

Test: twoExamples.info

Date: 2016-05-24 15:17:38

<table>
<thead>
<tr>
<th>Filename</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>evenExample.cpp</td>
<td>85.7%</td>
<td>100.0%</td>
</tr>
<tr>
<td>isEven.hpp</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>oddExample.cpp</td>
<td>85.7%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

Generated by: LCOV version 1.12-4-g04a3c0e

This is the file we’re testing
A simple example

We tested every line of this function.
A real example - xSDKTrilinos

- Part of the Trilinos library, developed at SNL as part of the IDEAS project
- Contains the interfaces between Trilinos, PETSc, and hypre
- Available at [https://github.com/trilinos/xSDKTrilinos](https://github.com/trilinos/xSDKTrilinos)
- Ten automated tests are run nightly
  - Six are actually examples that were converted into tests
- Did we leave anything out?
A real example - xSDKTrilinos

- Step 1: Modify our CMake configuration file to use the --coverage flag to compile and link

```
-D TPL_ENABLE_PETSC:BOOL=ON 
-D PETSC_LIBRARY_DIRS:FILEPATH="${PETSC_LIB_DIR}" 
-D PETSC_INCLUDE_DIRS:FILEPATH="${PETSC_INCLUDE_DIR}" 

-D TPL_ENABLE_ParMETIS:BOOL=ON 
-D ParMETIS_LIBRARY_DIRS:FILEPATH="${SUPERLU_LIB_DIR}" 
-D ParMETIS_INCLUDE_DIRS:FILEPATH="${SUPERLU_INCLUDE_DIR}" 

-D TPL_ENABLE_HYPRE:BOOL=ON 
-D HYPRE_LIBRARY_DIRS:FILEPATH="${HYPRE_LIB_DIR}" 
-D HYPRE_INCLUDE_DIRS:FILEPATH="${HYPRE_INCLUDE_DIR}" 

-D TPL_ENABLE_SuperLUDist:BOOL=ON 
-D SuperLUDist_LIBRARY_DIRS:FILEPATH="${SUPERLU_LIB_DIR}" 
-D SuperLUDist_INCLUDE_DIRS:FILEPATH="${SUPERLU_INCLUDE_DIR}" 

-D Trilinos_ENABLE_Amesos2:BOOL=ON 
-D Trilinos_ENABLE_xSDKTrilinos:BOOL=ON 

-D CMAKE_CXX_FLAGS:STRING="--coverage" 
-D CMAKE_C_FLAGS:STRING="--coverage" 
-D CMAKE_EXE_LINKER_FLAGS:STRING="--coverage" 
-D Trilinos_ENABLE_Fortran:BOOL=OFF 

${TRILINOS_HOME}
-- INSERT --
```
A real example - xSDKTrilinos

- Build Trilinos (including xSDKTrilinos)
  - ./do-configure
  - make -j
- This will create a whole bunch of .gcno files
- This will also build the xSDKTrilinos tests because the configure file included
  - -D Trilinos_ENABLE_TESTS:BOOL=ON
  - -D Trilinos_ENABLE_EXAMPLES:BOOL=ON
  - -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES=ON
A real example - xSDKTrilinos

- Run the tests using ctest
  - Note that this is not prohibitively slow
A real example - xSDKTrilinos

- All tests passed. Yay!
  - This also created a bunch of .gcda files

---

```
trilinos-build : ctest

Start 10: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
10/18 Test #10: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.42 sec
Start 11: xSDKTrilinos_PETSc_Anasazi_example_MPI_4
11/18 Test #11: xSDKTrilinos_PETSc_Anasazi_example_MPI_4 ............... Passed 2.71 sec
Start 12: xSDKTrilinos_PETSc_IFpack2_example_MPI_4
12/18 Test #12: xSDKTrilinos_PETSc_IFpack2_example_MPI_4 ............... Passed 1.47 sec
Start 13: xSDKTrilinos_PETSc_MueLu_example_MPI_4
13/18 Test #13: xSDKTrilinos_PETSc_MueLu_example_MPI_4 ............... Passed 2.34 sec
Start 14: xSDKTrilinos_example_TpetraKSP_MPI_4
14/18 Test #14: xSDKTrilinos_example_TpetraKSP_MPI_4 ............... Passed 1.50 sec
Start 15: xSDKTrilinos_example_EpetraKSP_MPI_4
15/18 Test #15: xSDKTrilinos_example_EpetraKSP_MPI_4 ............... Passed 1.37 sec
Start 16: xSDKTrilinos_HypreTest_MPI_4
16/18 Test #16: xSDKTrilinos_HypreTest_MPI_4 ............... Passed 1.42 sec
Start 17: xSDKTrilinos_Hypre_Belos_example_MPI_4
17/18 Test #17: xSDKTrilinos_Hypre_Belos_example_MPI_4 ............... Passed 1.38 sec
Start 18: xSDKTrilinos_Hypre_Solve_example_MPI_4
18/18 Test #18: xSDKTrilinos_Hypre_Solve_example_MPI_4 ............... Passed 1.36 sec

100% tests passed, 0 tests failed out of 18

Label Time Summary:
  Amoses2       =   12.67 sec (8 tests)
  xSDKTrilinos  =   16.39 sec (10 tests)

Total Test time (real) =   29.11 sec
```

[amklinv@s995692 trilinos-build]$
A real example - xSDKTrilinos

- Collect coverage data for the tests using lcov
  - `lcov --capture --directory . --output-file xSDKTrilinos.info`
  - This creates `xSDKTrilinos.info`
  - Lcov processes 634 gcda files in this step, so this does take a few minutes
A real example - xSDKTrilinos

- Generate html output using genhtml
  - genhtml xSDKTrilinos.info --output-directory xSDKTrilinos
  - This generates html files in the directory xSDKTrilinos
  - This step takes a few minutes too
A real example - xSDKTrilinos

**LCOV - code coverage report**

Current view: **top level - xSDKTrilinos/petsc/src**

- **Test:** xSDKTrilinos.info
- **Date:** 2016-06-02 15:36:10

<table>
<thead>
<tr>
<th>Filename</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BelosPETScSolMgr.hpp</td>
<td>84.7 %</td>
<td>68.2 %</td>
</tr>
<tr>
<td>Tpetra_PETScAIJGraph.hpp</td>
<td>75.3 %</td>
<td>62.5 %</td>
</tr>
<tr>
<td>Tpetra_PETScAIJMatrix.hpp</td>
<td>80.7 %</td>
<td>65.9 %</td>
</tr>
</tbody>
</table>

Generated by: [LCOV version 1.12-4-g04a3c0e](https://github.com/ tapsiha/LCOV)

Let’s take a look at the solver interface.
template<class ScalarType, class MV, class OP>

PetscErrorCode PETScSolMgr<ScalarType,MV,OP>::applyPrec(PC M, Vec x, Vec Mx)
{
    using Teuchos::RCP;
    typedef PETScSolMgrHelper<ScalarType,MV,OP> Helper;

    PetscErrorCode ierr;
    const PetscScalar * xData;
    PetscScalar * MxData;
    void * ptr;

    // Get the problem out of the context
    ierr = PCShellGetContext(M,&ptr); CHKERRQ(ierr);
    LinearProblem<ScalarType,MV,OP> * problem = (LinearProblem<ScalarType,MV,OP>*)ptr;

    // Rip the raw data out of the PETSc vectors
    ierr = VecGetArrayRead(x, &xData); CHKERRQ(ierr);
    ierr = VecGetArray(Mx, &MxData); CHKERRQ(ierr);

    // Wrap the PETSc data in a Trilinos Vector
    RCP<MV> trilinosX, trilinosMX;
    Helper::wrapVector(const_cast<PetscScalar*>(xData), *problem->getLHS(), trilinosX);
    Helper::wrapVector(MxData, *problem->getLHS(), trilinosMX);

    // Perform the multiplication
    if(problem->isLeftPrec()) {
        problem->applyLeftPrec(*trilinosX, *trilinosMX);
    } else {
        problem->applyRightPrec(*trilinosX, *trilinosMX);
    }

    // Unwrap the vectors; this is necessary if we copied data in the wrap step
    Helper::unwrapVector(MxData, trilinosMX);

    // Restore the PETSc vectors
    ierr = VecRestoreArrayRead(x,&xData); CHKERRQ(ierr);
    ierr = VecRestoreArray(Mx,&MxData); CHKERRQ(ierr);

    return 0;
}
A real example - xSDKTrilinos

```cpp
791  :   // Perform the multiplication
792 192  :   if(problem->isLeftPrec()) {
793 192  :     problem->applyLeftPrec(*trilinosX, *trilinosMX);
794  :   }
795  :   else {
796 0  :     problem->applyRightPrec(*trilinosX, *trilinosMX);
797  :   }
```

Oops. I never tested the RIGHT preconditioning branch.
Why is documentation important?

- To identify the purpose of the software and its requirements
- To clarify what each component does, what is needed to maintain it, and how it can be reused elsewhere
- To provide user support
  - Minimizes unnecessary handholding of users
- To ensure that software is used within its region of validity
  - Minimizes possibility of producing spurious scientific results
Categories of documentation

- Users guide
- Reference manual
  - List of the interfaces and routines and explanation of functionality
  - Can be generated automatically from code
- Readme files
- Installation guide
- Tutorials

All software needs documentation
Not all software needs a users guide
How does Trilinos handle documentation?

- Each package does it differently
- User manuals
  - MueLu (algebraic multigrid)
  - AztecOO (Krylov solvers)
  - Teuchos RCP (reference counted pointers)
- Publicly available tutorials, presentations, and slides
  - Tpetra (MPI+X linear algebra)
  - Kokkos*
- Well commented examples
- Automatically generated html documentation
Doxygen

- One approach to producing “reference manual”-like documentation
- Automatically generates html documentation from comments in source code
- Easy to update documentation when source code is updated
- doxywizard - GUI frontend for doxygen
A simple doxygen example

- Add some comments to isEven.hpp

```cpp
/**
 * @file isEven.hpp
 * Contains a function for detecting whether a number is even or odd
 * @author Alicia Klinvex
 * @example oddExample.cpp
 * @example evenExample.cpp
 */

/**
 * Detects whether an integer is even or odd
 * @param[in] x an integer which may be even or odd
 * @return
 * - true if x is even
 * - false otherwise
 */

bool isEven(int x)
{
    if(x%2 == 0)
        return true;
    return false;
}
```
A simple doxygen example

- Create an index page (index.doc)

```
/**!
 * \mainpage EvenOdd: a revolutionary new function
 * \section intro Introduction
 * This page provides the documentation for the %EvenOdd project.
 */
```
A simple doxygen example

Step 1: Specify the working directory from which doxygen will run
/home/amklinv/IDEAS/testingTalk/examples/simpleExample

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Provide some information about the project you are documenting

- **Project name:** EvenOdd
- **Project version or id:**

Specify the directory to scan for source code

- **Source code directory:** /home/amklinv/IDEAS/testingTalk/examples/simpleExample
- **Scan recursively**

Specify the directory where doxygen should put the generated documentation

- **Destination directory:** /IDEAS/testingTalk/examples/simpleExample/doxyFiles

Next
A simple doxygen example

Step 1: Specify the working directory from which doxygen will run
/home/amklinv/IDEAS/testingTalk/examples/simpleExample

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Select the output format(s) to generate

- **HTML**
  - plain HTML
  - with frames and a navigation tree
  - prepare for compressed HTML (.chm)
  - With search function (requires PHP enabled web server)

- **LaTeX**
  - as intermediate format for hyperlinked PDF
  - as intermediate format for PDF
  - as intermediate format for PostScript

- Man pages
- Rich Text Format (RTF)
- XML

Previous  Next
A simple doxygen example

Step 1: Specify the working directory from which doxygen will run

/home/amklinv/IDEAS/testingTalk/examples/simpleExample

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation.
A simple doxygen example

Step 1: Specify the working directory from which doxygen will run
/home/amklinv/IDEAS/testingTalk/examples/simpleExample

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation.

EXAMPLE_PATH
.

EXAMPLE_PATTERNS
*Example.cpp

EXAMPLE_RECURSIVE

IMAGE_PATH

Previous

Next
A simple doxygen example

Step 1: Specify the working directory from which doxygen will run
/home/amklinv/IDEAS/testingTalk/examples/simpleExample

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation.

Generating group documentation...
Generating group index...
Generating class documentation...
Generating annotated compound index...
Generating alphabetical compound index...
Generating hierarchical class index...
Generating member index...
Generating example index...
Generating namespace index...
Generating namespace member index...
Generating graph info page...
Generating file index...
Generating example index...
Generating file member index...
*** Doxygen has finished

Show HTML output
A simple doxygen example

EvenOdd: a revolutionary new function

Introduction

This page provides the documentation for the EvenOdd project.
A simple doxygen example

isEven.hpp File Reference

Go to the source code of this file.

Functions

bool isEven (int x)

Detailed Description

Contains a function for detecting whether a number is even or odd

Author:
Alicia Klinvex
A simple doxygen example

isEven.hpp

Go to the documentation of this file.

```cpp
bool isEven(int x) {
    if(x%2 == 0)
        return true;
    return false;
}
```
A simple doxygen example

Function Documentation

```cpp
/**
 * Detects whether an integer is even or odd
 * @param[in] x an integer which may be even or odd
 * @return
 * - true if x is even
 * - false otherwise
 */
```

bool isEven ( int x )

Detects whether an integer is even or odd

**Parameters:**

[in] x an integer which may be even or odd

**Returns:**

- true if x is even
- false otherwise

**Examples:**

evenExample.cpp, and oddExample.cpp.
Summary

- Testing and documentation are very important
- There are many different types of tests that should be included in your test suite
- Code coverage tools can help you figure out where existing testing is insufficient
- Documentation does not have to mean “user manual”
- Tools such as doxygen can help you write documentation
Thanks for Participating!

• Make sure you get counted. Please visit http://bit.ly/hpcbp-s04
• We want to improve this series. Please send feedback to HPCBestPractices+session04@gmail.com
• Slides and a recording will be available from the OLCF training web site: https://www.olcf.ornl.gov/training-event/webinar-series-best-practices-for-hpc-software-developers

Next Webinar

Session 5: How the HPC Environment is Different from the Desktop (and Why)
Date: Wednesday, July 14, 2016
Time: 1:00-2:00 pm ET
Presenter: Katherine Riley, Argonne Leadership Computing Facility


For updates, please register (if you haven’t already)
https://www.olcf.ornl.gov/training-event/webinar-series-best-practices-for-hpc-software-developers