PyTrilinos Users Guide

Bill Spotz
wfspotz@sandia.gov
Sandia National Laboratories

31 December 2010

Abstract

PyTrilinos is a python interface to selected Trilinos packages. The Trilinos Project is a collection of over 30 software packages written primarily in C++ that provide linear, nonlinear, and eigen solvers, along with preconditioners and supporting utilities, that are object-oriented, parallel and serial, for sparse and dense problems. PyTrilinos is one of those packages, and provides python interfaces to the most popular and important Trilinos packages. This Guide provides information for users just getting started with PyTrilinos.

Contents

Prereqresites **1**

Building and Installing **2**
Trilinos Packages 2
Python and SWIG 2
Installation Path 3
Build Time 3
Configuring and Building 3

Prereqresites

To build and execute PyTrilinos, you must have python installed. PyTrilinos is not yet ported to python 3, so you need python 2.X, where X is 5 or greater.

In addition, you need to have numpy installed relative to the same python installation. NumPy is numerical python, a python module for handling homogenous, contiguous arrays of data. Currently, version 1.0 or greater is required.

Finally, you need SWIG, the Simple Wrapper Interface Generator, a tool that reads C/C++ code and automatically generates python wrapper code. This is the workhorse for building PyTrilinos code. Version 2.0 or greater is required.

Building and Installing

PyTrilinos is a Trilinos package, and so is built by building Trilinos in the normal way and enabling the PyTrilinos package. Trilinos uses the CMake build system, which requires a build directory separate from the source directory. Trilinos builds tend to be complex enough that it is recommended that you write a short script to invoke `cmake` to configure the build. See the Trilinos documentation for more details.

PyTrilinos is not enabled by default. To enable PyTrilinos, use the `cmake` option:

```
-D Trilinos_ENABLE_PyTrilinos:BOOL=ON
```

PyTrilinos requires that all of Trilinos be built as shared libraries. This is not the default, nor does enabling PyTrilinos set this flag. You must therefore explicitly turn on shared libraries:

```
-D BUILD_SHARED_LIBS:BOOL=ON
```

If you enable PyTrilinos but forget to turn on shared libraries, `cmake` will exit and give you a gentle reminder to set this option.

Trilinos Packages

PyTrilinos will build python interfaces only for those Trilinos packages that are enabled. Some of the default packages have python wrappers, some do not. Some Trilinos packages with python wrappers are not in the default set. So not specifically enabling any other Trilinos packages (i.e., using the default set of packages) will build an incomplete PyTrilinos.

The simplest way to enable every Trilinos package that has a python interface is to use:

```
-D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON
```

All packages with python wrappers are seen as optional packages to PyTrilinos, so they all get enabled. But this directive is recursive, so those packages that are optional to PyTrilinos also have optional packages, and they get enabled as well. This approach will build several more packages than you need.

For complete control, you can use the option:

```
-D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF
```

and then individually enable every package you want using:

```
-D Trilinos_ENABLE_<PACKAGE>:BOOL=ON
```

where `<PACKAGE>` is the name of a Trilinos package. The Trilinos packages with python interfaces are Teuchos, Epetra, Triutils, EpetraExt, AztecOO, Galeri, Amesos, Ifpack, Komplex, Anasazi, ML and NOX.

Python and SWIG

You may specify the python interpreter (and thus the version of the Python/C API) with the `cmake` option:

```
-D PYTHON_EXECUTABLE:FILEPATH=...
```

If you do not specify this option, `cmake` actually looks for an executable named `python2.7` (and then `python2.6`, etc., down to `python1.5`, and then finally for `python`).

The SWIG executable my be similarly specified with:

```
-D SWIG_EXECUTABLE:FILEPATH=...
```

If this option is not used, `cmake` will find the first occurence of `swig` in your environment's path.

Installation Path

Because PyTrilinos is imported into a running `python` interpreter, and that interpreter will have specific places it looks for python modules, it is possible to set the PyTrilinos installation path separately from the Trilinos installation path. Use:

```
-D PyTrilinos_INSTALL_PREFIX:PATH=...
```

If this option is not specified, then the PyTrilinos installation path prefix is defined by the Trillinos installation path prefix:

```
-D CMAKE_INSTALL_PREFIX:PATH=...
```

If neither of these options is specified, then the PyTrilinos installation path is set to be the location where the `python` executable expects to find python modules. Since specifying the PyTrilinos installation location is more complex than for the rest of Trilinos, the PyTrilinos installation path is written to the screen near the end of the configuration process.

Build Time

Trilinos can take a long time to configure and build, depending on the number of packages that are enabled. A significant portion of this time can be taken up by building tests and examples. You can save a considerable amount of time by turning these tests and examples off:

```
-D Trilinos_ENABLE_TESTS:BOOL=OFF
-D Trilinos_ENABLE_EXAMPLES:BOOL=OFF
```

Configuring and Building

To configure Trilinos, run `cmake` with the desired options specified, followed by the path to the top Trilinos source directory. This will generate the build system. On a Unix machine, then run `make` to build all of the enabled Trilinos packages. Then run `make install` (or `sudo make install`, if installing to a path that requires root privileges) to install the Trilinos libraries, header files and python modules to their proper place.