How to Configure Software

The IDEAS Scientific Software Productivity Project

ideas-productivity.org/resources/howtos/

Overview: Most CSE software needs to be installed from source on a wide variety of machines by end users. Developers of the software must decide how to enable this installation easily without overburdening the developers of software or the end users. This document introduces several approaches to use depending on the contents and scale of the software package.

Target Audience: Scientific software project leaders and developers who need to ensure that their software can be installed on a wide variety of machines.

Prerequisites: First read the document *What Is Software Configuration?*

For simple packages that have almost no dependencies or machine dependent parameters, the use of (1) an options file is acceptable. For packages that incrementally build on another package, it can be reasonable to piggyback on the other package's configuration information, thus requiring the end user merely to edit a file to indicate the location of the piggybacked package.

For all other packages we recommend using the open-source tools (2) GNU *Autotools* or (3) *CMake* or (4) rolling your own configuration system that utilizes the syntax of the Autoconf command line arguments (see comparison of features in the table below). GNU Autotools is widely used and has a great deal of support resources on the web but is a bit idiosyncratic. Almost all Linux users and many Mac OS users are familiar with and expect the Autotools command line syntax. CMake is a product of a for-profit company *Kitware* whose business model is based on paid customer support (CMake is open-source software and free community support is available on the open CMake mail lists). CMake comes with a testing environment CTest (which posts to a web dashboard CDash) and a packaging system CPack.

Creating a GNU Autoconf configure script

The GNU open source package *Autotools* is distributed by GNU and is available for any modern system. In brief, the software developer creates a "configure.in" file, which contains all options that can be set and any tests that should be run, and writes makefile templates that will be populated with information from the given options. Next, the developer runs Autoconf to generate a "configure" file, which will then be included as part of the source distribution. Autotools also provides Automake, which helps with the generation of the makefiles from a Makefile.am file, and Libtool, which helps to manage building libraries. The details on how to make a usable "configure.in" file and makefile templates are beyond the scope of this document; numerous books and online references can help with these tasks. For high-performance computing machines that utilize a batch system – that is, require submitting all programs to a queue to be run on the machine – GNU Autotools can be problematic since it relies on being able to automatically build and run programs to determine machine properties.

Creating a CMake build

CMake has very different syntax from the GNU Autoconf configure scripts but performs essentially the same function. Thus, it has analogous options and variables that must be communicated to the build system. As an example, where configure may expect to set the C compiler by checking the passed in 'CC' variable, CMake looks for the option 'CMAKE_C_COMPILER' first and if it does not find it, uses the compiler listed by the environment variable 'CC'.

A CMake project must define a CMakeLists.txt file in its distribution. This file is analogous to the configure.in file for GNU Autoconf builds (and the Makefile.am used by Automake), in that it describes what options or variables are expected from the software installer and how those options will be broadcast to the makefiles that actually compile the software.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

Unlike GNU Autotools, however, CMake provides facilities for use on high-performance computing machines that utilize a batch system. Unfortunately, it relies on a database of machine properties, and this database is often out of date for many high-performance computing sites. GNU Autotools and CMake both support cross-compiling, which can be used on a batch-based system, but this disables the useful configure-time testing that the tools provide.

Rolling your own configure script

A “configure” script can also be written without using the GNU Autotools programs and not requiring the use of M4 for those who want to avoid it.[1] This approach has the advantage of allowing additional features and behavior not provided by GNU. It is highly recommended that any other configure script follow the syntax and expected behavior of a GNU Autoconf-generated script.

- Use the --help option to list and explain the various options available.
- Use --prefix= to denote where to install header files and binaries.
- Check the command line and the environment for compiler variables CC, CFLAGS, CXX, CXXFLAGS, and other similar variables. If these are not specifically given, then make some guesses based on what executables are available.

No matter how your configure script is created, users expect certain conventions.

- Use --with-xxx (and --without-xxx) for describing what is available in the install environment. For example, if your build behaves differently if LAPACK is available or not, then you should use a --with-lapack option.[2]
- Use --enable-xxx (and --disable-xxx) to turn on or off features in the code, such as options to build fortran interfaces, debugging symbols, or shared libraries.
- Print out useful messages if there is a problem or inconsistency with the options.

Comparison of Software Configuration Features

Feature	**(1) Parameter file**	**(2) GNU Autotools**	**(3) CMake**	**(4) Roll your own configure**
Can automatically determine machine parameters		Yes	Yes	Yes, but you must provide all these tests as part of your system
Requires a large initial investment		Yes	Yes	Yes, as you are starting from scratch you cannot leverage previously developed code
Has complex capabilities		Yes	Yes	Yes, but only if you write them
Can be modified as needed	Yes	Yes	Yes	Yes

[1]For example, PETSc’s configure, written completely in Python, provides all the functionality of GNU Autotools as well as the ability to install other packages and work on batch computer systems.

[2]GNU recommends that you do not use locations with these variables but instead add any necessary flags to the LDFLAGS environment variable (i.e., GNU prefers LDFLAGS=/usr/lib/libpack.a --with-lapack instead of --with-lapack-dir=/usr/lib/liblapack.a). We disagree with this recommendation because the latter approach allows tests to be written for each particular option.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

Feature	**(1) Parameter file**	**(2) GNU Autotools**	**(3) CMake**	**(4) Roll your own configure**
Works with Microsoft Visual Studio	User must manually enter required information into Visual Studio. This is painful and error prone.		Yes, generates native Microsoft Visual Studio project files	Yes, but you must provide the appropriate code to generate the needed files
Native windows builds (that may then be used from within Microsoft Visual Studio)			Yes, generates native NMake and Ninja build files for all native Windows compilers	Yes, but you have to write all of such support by yourself (i.e., can use Makefiles.)
Requires learning new scripting language		Yes	Yes	
Can test given options for compatibility		Yes	Yes	Yes, but you must provide the tests.
Is commonly used, documentation available		Yes	Yes	
Works with IDE's	User must manually enter required information into the IDE. This is painful and error prone.		Yes, generates native project files for XCode, Eclipse, etc.	Yes, but you must provide the appropriate code to generate the needed files
Supports different backend build tools other than Makefiles (e.g. Ninja, NMake)	User must manually enter required information. This is generally painful and error prone.		Yes	Yes, but you must write the tools to generate such backend build files
Portable support for shared libraries		Yes, but you need to also use libtool or gmake as well.	Yes	Yes, but you need to provide the knowledge and logic for every platform
Portable automatic generation of dependency information		Yes, but requires usage of automake and requires compiler support for generating dependency info, which they all have	Yes, built into the CMake executable, not dependent on compiler support	Yes, but you have to roll your own
Provides database for known HPC computer systems			Yes, though sometimes not available for new prototype systems	
Includes graphical interface			Yes	

Regardless of the method used to configure, your system should do the following.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

- Provide end users access to the options that were used to configure the file and also the internal variables that were set when the "configure" script ran. This information is helpful for reconfiguring a software package to prevent having to duplicate any trial-and-error learning process, and it can help keep all dependencies consistent for future software builds.
- Make proper documentation available for the installer. At the least, any required software dependencies should be listed, every configuration variable should be explained, and appropriate examples should be given. This documentation can be on an installation instruction web page or text file included in the distribution, and it should also be available from command line help queries.

Common configuration options for scientific software

Many scientific software libraries and applications require the same information when configuring, such as locations of BLAS and LAPACK. When a set of interacting software libraries is built, the same options must be used for each of the libraries. In the past each software package selected its own name for these options, making installing multiple packages painful and error prone. The IDEAS team has developed a set of *standard configuration options* that we recommend you follow.

This document was prepared by Jason Sarich with key contributions from Roscoe Bartlett, Michael A. Heroux, Barry Smith, and James M. Willenbring.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.