



Draft document generated by the IDEAS xSDK project.

We are actively soliciting suggestions from the community at github.com/xsdk-project/xsdk-community-policies.

xSDK Community Installation Policies

Version 0.5.0, June 27, 2019

<https://xsdk.info/policies>

Background

What is software configuration? and How to configure software.

Motivation

Combinations of multiple software packages developed by different groups have become essential for large-scale computational science, where the capabilities needed for modeling, simulation, and analysis are broader than any single team has resources to address. The often-tedious trial-and-error process of obtaining, configuring, and installing any single tool may arguably be manageable. From the perspective of an end-user application scientist, however, handling each tool's installation idiosyncrasies can easily become overwhelming when dealing with several packages in combination. Worse, such problems are compounded by the need for consistency among packages to be used within the same application in terms of compiler, compiler version, exotic compiler optimization options, and common third-party packages such as BLAS and HDF5.

Goal

A key aspect of work in the IDEAS software productivity project is developing an Extreme-scale Scientific Software Development Kit (xSDK). As an initial step in this work, our goal is to define and implement a standard subset¹ of configure and CMake² options for xSDK and other HPC packages in order to make the configuration and installation process as efficient as possible on standard Linux distributions and Mac OS, as well as on target machines at DOE computing facilities (ALCF, NERSC, OLCF). Note that we are not requiring

that all packages use the same installation software, merely that they follow the same standard procedure with the same option names for installation. This approach provides maximum flexibility for each package to select the most suitable toolchain to use for its package.

Impact

Development of a standard xSDK package installation interface is a foundational step toward the seamless combined use of multiple xSDK libraries. The impact of this work is that all xSDK packages will have standard configuration and build instructions, as well as a tester to ensure that all functionality works properly. In addition, because new packages in the xSDK will follow the same standard, it is possible to make the installations “scriptable,” that is, to write tools to install many packages automatically. This work is part of the xSDK Community Package Policies.

xSDK Standard Configure and CMake Options³

- I1.** Implement the xSDK defaults option.
- I2.** Identify location to install package.
- I3.** Select compilers and compiler flags.
- I4.** Create libraries with debugging information and possible additional error checking.
- I5.** Select option used for indicating whether to build shared libraries.
- I6.** Build interface for a particular additional language.
- I7.** Determine precision for packages that build only for one precision. Packages that handle all precisions automatically are free to ignore this option.
- I8.** Determine index size for packages that build only for one index size. Packages that handle all precisions automatically are free to ignore this option.
- I9.** Set location of BLAS and LAPACK libraries.
- I10.** Determine other package libraries and include directories.
- I11.** In the XSDK mode, XSDK projects should not rely on users providing any library path information in environmental variables.
- I12.** Provide commands for compiling, installing, and “smoke” testing.
- I13.** Package should provide a machine-readable output to show provenance.

For further discussion and examples of the xSDK standard Configure and CMake options see `discussions_and_examples.md` .

- Changes in version 0.5.0, June 27, 2019:
- Changed installation policies 13 and 10 and examples in 10

Authors

This document was prepared by Roscoe Bartlett, Jason Sarich, and Barry Smith, with key input from Todd Gamblin. We thank xSDK software developers and the IDEAS team for insightful discussion about issues and approaches.

Acknowledgement

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.

¹ Packages are free to support their own additional options, but using the standard options should be all that is needed to get correct builds.

² A subset of these standard behaviors is implemented in the XSDKDefaults.cmake module and is demonstrated and tested in the CMake project <https://github.com/bartlettroscoe/XSDKCMakeProj>.

³ This standard is related only to arguments to CMake and GNU Autoconf; there is no requirement regarding the make system used (for example, that it be GNU make) nor that the make system accepts any particular arguments, such as `make LIBS+=-lz`.

I1

Implement the default behavior described below. Each package can decide whether XSDK mode is the default mode.⁴

1. `--dis/enable-xsdk-defaults`
2. `USE_XSDK_DEFAULTS=[YES,NO]`

⁴ For packages like Trilinos that need to maintain backward compatibility over consecutive releases, `USE_XSDK_DEFAULTS` may be `FALSE` by default.

I2

Identify location to install package. Multiple “versions” of packages, such as debug and release, can be installed by using different prefix directories.

1. `--prefix=directory`
2. `CMAKE_INSTALL_PREFIX=directory`

I3

Select compilers⁵ and compiler flags.

1. If the compilers (and or flags) are explicitly set on input, use those:
 1. `CC=<cc>`, `CXX=<cxx>`, `FC=<fc>`, `CPP`, `FFLAGS`, `FCFLAGS`, `CFLAGS`, `CXXFLAGS`, `CPPFLAGS`, `LDFLAGS`
 2. `CMAKE_C_COMPILER=<cc>`, `CMAKE_CXX_COMPILER=<cxx>`, `CMAKE_Fortran_COMPILER=<fc>`, `CMAKE_C_FLAGS=<flag1> <flag2> ...`, `CMAKE_CXX_FLAGS=<...>`, `CMAKE_Fortran_FLAGS=<...>`
2. If the compilers and/or flags are not explicitly set on input but are set in the environment variables `FC`, `CC`, `CXX`, `CPP`⁶, `FFLAGS`, `FCFLAGS`, `CFLAGS`, `CXXFLAGS`, `CPPFLAGS`,⁷ `LDFLAGS`, then the compilers and flags must be set to these. If both `FFLAGS` and `FCFLAGS` are set, then they need to be the same or it is an error.
3. If the compilers and/or compiler flags are not explicitly passed in or defined in the environment variables listed above, then the package is free to try to find compilers on the system and set the compiler flags consistent with the other settings defined below (e.g., shared libraries vs. static libraries, debug vs. non-debug).⁸

⁵ Packages must support using the MPI compiler wrappers for these arguments.

⁶ The environment variable `CPP` is not supported by raw CMake.

⁷ The environment variable `CPPFLAGS` is not supported by raw CMake.

⁸ All CMake projects should use the same built-in CMake algorithm to find the default compilers, so even when no explicit compilers or flags are set they should use the same compilers and flags. Also, raw CMake projects will append compiler flags based on the build type. See “Selecting compiler and linker options”.

I4

Create libraries with debugging information and possible additional error checking (default is debug in XSDK mode).

1. `--dis/enable-debug`
2. `CMAKE_BUILD_TYPE=[Debug,Release]`

I5

Select option used for indicating whether to build shared libraries (default is shared in XSDK mode).

1. `--dis/enable-shared (configure)`
2. `BUILD_SHARED_LIBS=[YES,NO]`

I6

Build interface for a particular additional language.

1. `--dis/enable-<language>`
2. `XSDK_ENABLE_<language>=[YES,NO]`

I7

Determine precision for packages that build only for one precision (default is double). Packages that handle all precisions automatically are free to ignore this option.

1. `--with-precision=[single,double,quad]`
2. `XSDK_PRECISION=[SINGLE,DOUBLE,QUAD]`

I8

Determine index size for packages that build only for one index size (default is 32). Packages that handle all index sizes automatically are free to ignore this option.

1. `--with-index-size=[32,64]`

2. `XSDK_INDEX_SIZE=[32,64]`

I9

Set location of BLAS and LAPACK libraries (default is to locate one on the system automatically).

1. `--with-blas-lib="linkable list of libraries" --with-lapack-lib="linkable list of libraries"` – it is fine to use `-L` and `-l` flags in the lists
2. `TPL_BLAS_LIBRARIES="linkable list of libraries", TPL_LAPACK_LIBRARIES="linkable list of libraries"` – should not use `-L` or `-l` flags in the lists

I10

Determine other package libraries and include directories.

1. `--with-<package> --with-<package>-lib="linkable list of libraries" --with-<package>-include="list of include directories"`
2. `TPL_ENABLE_<package>=[YES,NO]`

Packages must provide a way for a user to specify a dependent package to use. Packages are free to locate a package on the file system if none is specifically provided by the user. If the user does provide one, however, it must be used; if it is not able to be used, then an error must be generated. A package cannot silently substitute a different installation.

I11

In the XSDK mode, XSDK projects should not rely on users providing any library path information in environmental variables such as `LD_LIBRARY_PATH`.

I12

After packages are configured, they can be compiled, installed and “smoke” tested with the following commands: `make ; [sudo] make install ; make test_install`.