FindBoost

Find `Boost` include dirs and libraries

Use this module by invoking `find_package()` with the form:

```
find_package(Boost
  [version] [EXACT]      # Minimum or EXACT version e.g. 1.67.0
  [REQUIRED]             # Fail with error if Boost is not found
  [COMPONENTS <libs>...] # Boost libraries by their canonical name
                         # e.g. "date_time" for "libboost_date_time"
  [OPTIONAL_COMPONENTS <libs>...]
                         # Optional Boost libraries by their canonical name)
                         # e.g. "date_time" for "libboost_date_time"
)
```

This module finds headers and requested component libraries OR a CMake package configuration file provided by a "Boost CMake" build. For the latter case skip to the Boost CMake section below.

New in version 3.7: `bzip2` and `zlib` components (Windows only).

New in version 3.11: The `OPTIONAL_COMPONENTS` option.

New in version 3.13: `stacktrace_*` components.

New in version 3.19: `bzip2` and `zlib` components on all platforms.

Result Variables

This module defines the following variables:

`Boost_FOUND`
True if headers and requested libraries were found.

`Boost_INCLUDE_DIRS`
Boost include directories.

`Boost_LIBRARY_DIRS`
Link directories for Boost libraries.

`Boost_LIBRARIES`
Boost component libraries to be linked.

`Boost_<COMPONENT>_FOUND`
True if component `<COMPONENT>` was found (`<COMPONENT>` name is upper-case).

`Boost_<COMPONENT>_LIBRARY`
Libraries to link for component `<COMPONENT>` (may include `target_link_libraries()` debug/optimized keywords).

`Boost_VERSION_MACRO`
`BOOST_VERSION` value from `boost/version.hpp`.

`Boost_VERSION_STRING`
Boost version number in `X.Y.Z` format.

`Boost_VERSION`
Boost version number in `X.Y.Z` format (same as `Boost_VERSION_STRING`).

Changed in version 3.15: In previous CMake versions, this variable used the raw version string from the Boost header (same as `Boost_VERSION_MACRO`). See policy `CMP0093`.

`Boost_LIB_VERSION`
Version string appended to library filenames.

`Boost_VERSION_MAJOR`, `Boost_MAJOR_VERSION`
Boost major version number (`X` in `X.Y.Z`).

`Boost_VERSION_MINOR`, `Boost_MINOR_VERSION`
Boost minor version number (`Y` in `X.Y.Z`).

`Boost_VERSION_PATCH`, `Boost_SUBMINOR_VERSION`
Boost subminor version number (`Z` in `X.Y.Z`).

`Boost_VERSION_COUNT`
Amount of version components (3).

`Boost_LIB_DIAGNOSTIC_DEFINITIONS` (Windows-specific)
Pass to `add_definitions()` to have diagnostic information about Boost's automatic linking displayed during compilation

New in version 3.15: The `Boost_VERSION_<PART>` variables.

Cache variables

Search results are saved persistently in CMake cache entries:

`Boost_INCLUDE_DIR`
Directory containing Boost headers.

`Boost_LIBRARY_DIR_RELEASE`
Directory containing release Boost libraries.

`Boost_LIBRARY_DIR_DEBUG`
Directory containing debug Boost libraries.

`Boost_<COMPONENT>_LIBRARY_DEBUG`
Component `<COMPONENT>` library debug variant.

`Boost_<COMPONENT>_LIBRARY_RELEASE`
Component `<COMPONENT>` library release variant.

New in version 3.3: Per-configuration variables `Boost_LIBRARY_DIR_RELEASE` and `Boost_LIBRARY_DIR_DEBUG`.

Hints

This module reads hints about search locations from variables:

`BOOST_ROOT`, `BOOSTROOT`
Preferred installation prefix.

`BOOST_INCLUDEDIR`
Preferred include directory e.g. `<prefix>/include`.

`BOOST_LIBRARYDIR`
Preferred library directory e.g. `<prefix>/lib`.

`Boost_NO_SYSTEM_PATHS`
Set to `ON` to disable searching in locations not specified by these hint variables. Default is `OFF`.

`Boost_ADDITIONAL_VERSIONS`
List of Boost versions not known to this module. (Boost install locations may contain the version).

Users may set these hints or results as `CACHE` entries. Projects should not read these entries directly but instead use the above result variables. Note that some hint names start in upper-case `BOOST`. One may specify these as environment variables if they are not specified as CMake variables or cache entries.

This module first searches for the Boost header files using the above hint variables (excluding `BOOST_LIBRARYDIR`) and saves the result in `Boost_INCLUDE_DIR`. Then it searches for requested component libraries using the above hints (excluding `BOOST_INCLUDEDIR` and `Boost_ADDITIONAL_VERSIONS`), "lib" directories near `Boost_INCLUDE_DIR`, and the library name configuration settings below. It saves the library directories in `Boost_LIBRARY_DIR_DEBUG` and `Boost_LIBRARY_DIR_RELEASE` and individual library locations in `Boost_<COMPONENT>_LIBRARY_DEBUG` and `Boost_<COMPONENT>_LIBRARY_RELEASE`. When one changes settings used by previous searches in the same build tree (excluding environment variables) this module discards previous search results affected by the changes and searches again.

Imported Targets

New in version 3.5.

This module defines the following `IMPORTED` targets:

`Boost::boost`
Target for header-only dependencies. (Boost include directory).

`Boost::headers`
New in version 3.15: Alias for `Boost::boost`.

`Boost::<component>`
Target for specific component dependency (shared or static library); `<component>` name is lower-case.

`Boost::diagnostic_definitions`
Interface target to enable diagnostic information about Boost's automatic linking during compilation (adds `-DBOOST_LIB_DIAGNOSTIC`).

`Boost::disable_autolinking`
Interface target to disable automatic linking with MSVC (adds `-DBOOST_ALL_NO_LIB`).

`Boost::dynamic_linking`
Interface target to enable dynamic linking with MSVC (adds `-DBOOST_ALL_DYN_LINK`).

Implicit dependencies such as `Boost::filesystem` requiring `Boost::system` will be automatically detected and satisfied, even if system is not specified when using `find_package()` and if `Boost::system` is not added to `target_link_libraries()`. If using `Boost::thread`, then `Threads::Threads` will also be added automatically.

It is important to note that the imported targets behave differently than variables created by this module: multiple calls to `find_package(Boost)` in the same directory or sub-directories with different options (e.g. static or shared) will not override the values of the targets created by the first call.

Other Variables

Boost libraries come in many variants encoded in their file name. Users or projects may tell this module which variant to find by setting variables:

`Boost_USE_DEBUG_LIBS`
New in version 3.10.

Set to `ON` or `OFF` to specify whether to search and use the debug libraries. Default is `ON`.

`Boost_USE_RELEASE_LIBS`
New in version 3.10.

Set to `ON` or `OFF` to specify whether to search and use the release libraries. Default is `ON`.

`Boost_USE_MULTITHREADED`
Set to OFF to use the non-multithreaded libraries ("mt" tag). Default is `ON`.

`Boost_USE_STATIC_LIBS`
Set to ON to force the use of the static libraries. Default is `OFF`.

`Boost_USE_STATIC_RUNTIME`
Set to `ON` or `OFF` to specify whether to use libraries linked statically to the C++ runtime ("s" tag). Default is platform dependent.

`Boost_USE_DEBUG_RUNTIME`
Set to `ON` or `OFF` to specify whether to use libraries linked to the MS debug C++ runtime ("g" tag). Default is `ON`.

`Boost_USE_DEBUG_PYTHON`
Set to `ON` to use libraries compiled with a debug Python build ("y" tag). Default is `OFF`.

`Boost_USE_STLPORT`
Set to `ON` to use libraries compiled with STLPort ("p" tag). Default is `OFF`.

`Boost_USE_STLPORT_DEPRECATED_NATIVE_IOSTREAMS`
Set to ON to use libraries compiled with STLPort deprecated "native iostreams" ("n" tag). Default is `OFF`.

`Boost_COMPILER`
Set to the compiler-specific library suffix (e.g. `-gcc43`). Default is auto-computed for the C++ compiler in use.

Changed in version 3.9: A list may be used if multiple compatible suffixes should be tested for, in decreasing order of preference.

`Boost_LIB_PREFIX`
New in version 3.18.

Set to the platform-specific library name prefix (e.g. `lib`) used by Boost static libs. This is needed only on platforms where CMake does not know the prefix by default.

`Boost_ARCHITECTURE`
New in version 3.13.

Set to the architecture-specific library suffix (e.g. `-x64`). Default is auto-computed for the C++ compiler in use.

`Boost_THREADAPI`
Suffix for `thread` component library name, such as `pthread` or `win32`. Names with and without this suffix will both be tried.

`Boost_NAMESPACE`
Alternate namespace used to build `boost` with e.g. if set to `myboost`, will search for `myboost_thread` instead of `boost_thread`.

Other variables one may set to control this module are:

`Boost_DEBUG`
Set to `ON` to enable debug output from `FindBoost`. Please enable this before filing any bug report.

`Boost_REALPATH`
Set to `ON` to resolve symlinks for discovered libraries to assist with packaging. For example, the "system" component library may be resolved to `/usr/lib/libboost_system.so.1.67.0` instead of `/usr/lib/libboost_system.so`. This does not affect linking and should not be enabled unless the user needs this information.

`Boost_LIBRARY_DIR`
Default value for `Boost_LIBRARY_DIR_RELEASE` and `Boost_LIBRARY_DIR_DEBUG`.

`Boost_NO_WARN_NEW_VERSIONS`
New in version 3.20.

Set to `ON` to suppress the warning about unknown dependencies for new Boost versions.

On Visual Studio and Borland compilers Boost headers request automatic linking to corresponding libraries. This requires matching libraries to be linked explicitly or available in the link library search path. In this case setting `Boost_USE_STATIC_LIBS` to `OFF` may not achieve dynamic linking. Boost automatic linking typically requests static libraries with a few exceptions (such as `Boost.Python`). Use:

```
add_definitions(${Boost_LIB_DIAGNOSTIC_DEFINITIONS})
```

to ask Boost to report information about automatic linking requests.

Examples

Find Boost headers only:

```
find_package(Boost 1.36.0)
if(Boost_FOUND)
  include_directories(${Boost_INCLUDE_DIRS})
  add_executable(foo foo.cc)
endif()
```

Find Boost libraries and use imported targets:

```
find_package(Boost 1.56 REQUIRED COMPONENTS
             date_time filesystem iostreams)
add_executable(foo foo.cc)
target_link_libraries(foo Boost::date_time Boost::filesystem
                          Boost::iostreams)
```

Find Boost Python 3.6 libraries and use imported targets:

```
find_package(Boost 1.67 REQUIRED COMPONENTS
             python36 numpy36)
add_executable(foo foo.cc)
target_link_libraries(foo Boost::python36 Boost::numpy36)
```

Find Boost headers and some *static* (release only) libraries:

```
set(Boost_USE_STATIC_LIBS        ON)  # only find static libs
set(Boost_USE_DEBUG_LIBS         OFF) # ignore debug libs and
set(Boost_USE_RELEASE_LIBS       ON)  # only find release libs
set(Boost_USE_MULTITHREADED      ON)
set(Boost_USE_STATIC_RUNTIME     OFF)
find_package(Boost 1.66.0 COMPONENTS date_time filesystem system ...)
if(Boost_FOUND)
  include_directories(${Boost_INCLUDE_DIRS})
  add_executable(foo foo.cc)
  target_link_libraries(foo ${Boost_LIBRARIES})
endif()
```

Boost CMake

If Boost was built using the boost-cmake project or from Boost 1.70.0 on it provides a package configuration file for use with find_package's config mode. This module looks for the package configuration file called `BoostConfig.cmake` or `boost-config.cmake` and stores the result in `CACHE` entry `Boost_DIR`. If found, the package configuration file is loaded and this module returns with no further action. See documentation of the Boost CMake package configuration for details on what it provides.

Set `Boost_NO_BOOST_CMAKE` to `ON`, to disable the search for boost-cmake.