Build and Install of Boost on WindowsXP[1] for use with Visual Studio .NET 2003

Prerequisites

Boost jam

Boost jam must be installed and made available. See the install instructions for *boost jam*.

Some current version of Python is recommended to be installed prior to the installation of boost. My Python version was v.2.3.

Extract archive

After *boost jam* was installed and bjam.exe was made available, *boost_1_33_1.tar.gz* must be extracted into some directory, e.g. *G:\DevelopMSVSnet\EVnetW32MSVC7*.

Set PATH variable and VC71_ROOT variable

If not already done, add the path to the *Visual Studio .NET 2003* binaries directory to the PATH variable, e.g. add C:\Programme\MSVS2003\Vc7\bin to PATH.

Dann ";C:\Programme\MSVS2003\Vc7\bin" anfügen und mit OK beenden.

To set the *VC71_ROOT* variable, use ->New, enter VC71_ROOT for the name and C:\Programme\MSVS2003\Vc7 for the value of the new variable (Notice that VC71_ROOT points to the installation directory of the Visual Studio .NET 2003 toolset. *Path* poits to the bin subdirectory, however).

To build the Boost.Python libraries, other three variables must be set. Namely, in case of my personal Python v.2.3. distribution, I had to set:

Variable	**Value**
PYTHON_ROOT	C:\Python23
PYTHON_VERSION	2.3
PYTHON_LIB_PATH	C:\Python23\libs

Boost Installation

The common build and install process is driven by the top-level build file (Jamfile)

Open a DOS command window and change to the directory where you have extracted the Boost distribution you downloaded, e.g. G:\DevelopMSVSnet\EVnetW32MSVC7\boost-1.31.0.

[1] This documentation was made for installation of boost on German version of Windows XP Professional. Hope, this causes no confusion

The default build and install attempts to build all available libraries and installs the libraries and Boost header files to default location C:\Boost. Within those directories libraries are installed to the "lib" subdirectory, and headers to an "include/boost-1_31" subdirectory, the version will reflect the distribution you are installing.

Invoke the build system, specifying the *Visual Studio .NET 2003* toolset to build and install. To use Boost with *Visual Studio .NET 2003*, the build command is:

```
bjam "-sTOOLS=vc-7_1" install
```

Then, wait patiently until the build process is finished. This may take quite some while, even on a 3 GHz PC.

Note that only some static and dynamic link libraries will be build. Other Boost utilities can be used/invoked directly by including the required header files.

There are a lot of - IMO quite advanced - options to control the build process. Have a look at the following table, if you have some specific needs for the build process.

Action	
none	Only builds the Boost libraries. This lets you do the first part of what the install action normally does without copying the built libraries to the install location.
install	Builds and installs Boost libraries and headers.
stage	Builds the Boost libraries and copies them into a common directory.
Option	
--help	Shows a short summary of the options and syntax of the command.
-sTOOLS=<*toolsets*>	The list of tools to compile with. Usually only one is needed.
--prefix=PREFIX	Install architecture independent files here. Default; C:\Boost on Win32. Default; /usr/local on Unix. Linux, etc.
--exec-prefix=EPREFIX	Install architecture dependent files here. Default; PREFIX
--libdir=DIR	Install libraries here. Default; EPREFIX/lib
--includedir=DIR	Install source headers here. The Boost headers are installed in a version specific "boost-<version>" subdirectory in this directory. Default; PREFIX/include
--builddir=DIR	Build in this location instead of building within the distribution tree. This moves where the sources for the libraries are compiled to before they are installed. Recommended!

Action	
--stagedir=DIR	When staging only, with the "stage" action, copy to the given location. Default; ./stage
--without-<*library*>	Do not build, stage, or install the specified library.
--with-<*library*>	Build, stage, or install the specified library. This changes the default from trying to build all possible libraries, to only building the specified libraries.
--with-python-root[=PYTHON_ROOT]	Build Boost.Python libraries with the Python devel packages located at PYTHON_ROOT. The Boost.Python libraries are built only if the build can find the Python development package at this location. Default; C:\Python24 on Win32. Default; /usr on Unix, Linux, Cygwin, etc.
--with-python-version[=2.4]	Build Boost.Python libraries with the Python version indicated. Default; 2.4.
--with-pydebug	Build Boost.Python libraries using the Python debug runtime. This builds an additional set of libraries for use with the debug version of Python. The regular versions of the Boost.Python libraries are also built.
-sHAVE_ICU=1	Build Boost.Regex libraries with Unicode support provided by the ICU libraries. ICU must have been built with the same compiler that you are using to build Boost, and must be installed into your compiler's include and library search paths. See the Boost.Regex installation documentation for more information.
-sICU_PATH=*path*	Build Boost.Regex libraries with Unicode support provided by the ICU libraries. ICU must have been built with the same compiler that you are using to build Boost, and must have been built (or installed to) directory *path*. For example if you configured ICU with --prefix=/usr/local/icu/3.3, then use -sICU_PATH=/usr/local/icu/3.3. See the Boost.Regex installation documentation for more information.
-sNO_COMPRESSION=1	Build Boost.Iostreams without support for the compression filters which rely on the non-Boost libraries zlib and libbz2. If you use Windows, this option is enabled by default. If you use UNIX, the compression filters will likely work with no configuration, so this option should not be necessary. For full details see Boost.Iostreams Installation.

There are additional options as supported by Boost.Build and Boost.Jam. Of the additional options perhaps the most imporant is "-sBUILD=<features/variants>" which lets you override what is built by default. The "<features/variants>" value is a list, separated by spaces, of build requests. Features take the form of a tag and a value or

values. And variants are single symbolic names for a collection of features. For example the default is to request "debug release <runtime-link>static/dynamic <threading>single/multi", in which "debug" and "release" are variants, and the rest features with two values each.

In case of interest in more details, see the full documentation at www.boost.org.

Have fun,

Dietmar, 20.12.2005