3rdparty libraries

Jelle Hellings

June 15, 2011

1 Introduction

This document describes how the `3rdparty` subdirectory of the `exbisim` solution directory is organized to allow the `exbisim` solution to build debug and release builds for 32bit and 64bit x86 target platforms.

2 The 3rd party libraries

This software project depends on several 3rd party libraries; we have used the following libraries:

1. Boost C++ libraries; version 1.46.1; retrieved from SourceForge.
2. libxml2; version 2.7.8; retrieved from xmlsoft.org.
3. STXXL: Standard Template Library for Extra Large Data Sets; version 1.3.1; retrieved from SourceForge.

Our software is developed and tested with Visual Studio 2010 SP1; using the Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.40219.01 for 80x86 and the Microsoft (R) C/C++ Optimizing Compiler Version 16.00.40219.01 for x64 compilers. The source code is not tested on other platforms; but we did keep platform depended code to a minimum. All platform dependent code is placed in files with `_pd` suffix. See the code guidelines for more details on file naming conventions used by this project.

3 Layout of the `3rdparty` subdirectory

The `3rdparty` subdirectory has a subdirectory `include` containing all header files for the 3rd party libraries. For each library n there is also a subdirectory `n` containing the build object files where our programs link with. The layout of these subdirectories `n` is `platform/config/lib` where `platform` is `win32` or `x86` and `config` is `debug` or `release`. For each possibility the libraries should be placed in the correct subfolder; the project and solution settings of the `exbisim` solution depend on this structure for correct building the various projects (and programs) for the various target configurations.

4 Setting up new Visual Studio Projects

The file `template.vcxproj.template` in the root directory of the `exbisim` solution can be used as a template of new projects. This template has all libraries, headers and other options preconfigured such that the project can successfully use the mentioned 3rd party libraries.

5 Building the libraries

In general we have followed the details in the documentation of each library when building the libraries. The following subsections describes details for each library.

5.1 Boost C++ libraries

See the documentation for details on the building process. We have written a script `build.bat` to build all variants. This script has a single parameter; the target where all resulting libraries are stored. See the script for any details.

5.2 STXXL

See the documentation for details on the building process. STXXL can only be build when the Boost C++ libraries have been build. The build process for STXXL has not been automated by us.

Use the `vcvarsall.bat` script provided by your Visual C++ distribution to set the build environment (x64 for 64bits, x86 or no argument for win32). Create the file `make.settings.local` in the root directory of the STXXL sources to configure the resulting STXXL library. Use the provided `stxxl.txt` as a starting point. In the root directory of the STXXL sources run `nmake library_msvc` to built the STXXL library; and move the resulting file `lib/libstxxl.lib` to the correct position in the `3rdparty` subdirectory. Clean up the remainder by running `nmake clean_msvc`.

5.3 libxml2

See the documentation in the subdirectory `win32/readme.txt` in the source distribution of libxml2 for details on building libxml2. Due to a bug in the 2.7.8 release we could not build libxml2 with the provided Makefile.msvc. Therefore we have replaced the not-working Makefile.msvc of the 2.7.8 release by a working version (file is retrieved from the GNOME source repository (commit ae874211d4c4cd1044d9fe5d598049a99526822b)). This working version can be found in the `3rdparty` subdirectory.

Use the `vcvarsall.bat` script provided by your Visual C++ distribution to set the build environment (x64 for 64bits, x86 or no argument for win32). Use the script `win32/configure.js` to configure the build you want. We only need

a very small portion of the libxml2 library (the XML Reader API); as such we have build with the following configuration:

```
cscript configure.js trio=no ftp=no http=no html=no c14n=no
                     catalog=no docb=no xpath=no xptr=no
                     xinclude=no iconv=no icu=no iso8859x=no
                     zlib=no xml_debug=no mem_debug=no
                     run_debug=no regexps=no modules=no
                     tree=no reader=yes writer=no walker=no
                     pattern=no push=yes valid=no sax1=no
                     legacy=no output=yes schemas=no
                     schematron=no python=no compiler=msvc
                     debug=no static=yes cruntime=[/MDd|/MD]
```

Where `cruntime=/MDd` is used for debug builds and `cruntime=/MD` is used for release builds. After configuration we have run `nmake /f all` and `nmake /f install` for building the library. The resulting directories win32/include and win32/lib are used for building our applications; and thus are moved to the appropriate subdirectories of the `3rdparty` subdirectory of the `exbisim` solution. After building one can clean up by running `nmake /f clean`.