## The following copied from online instructions for building libSMBL at

<http://sbml.org/Software/libSBML/docs/cpp-api/libsbml-installation.html>

## 3. Procedures for Windows

There are two ways to compile libSBML under Windows: using the native Windows compilation tools from Microsoft, and using the [Cygwin](http://cygwin.org/" \t "_blank) environment. In this section, we focus on using the native Windows environment because this appears to be the more popular approach used by Windows-based users of libSBML. Cygwin users can follow essentially the same [instructions as for other Unix environments](http://sbml.org/Software/libSBML/docs/cpp-api/libsbml-installation.html#basic-config) given above.

### *3.1 Configuring using CMake*

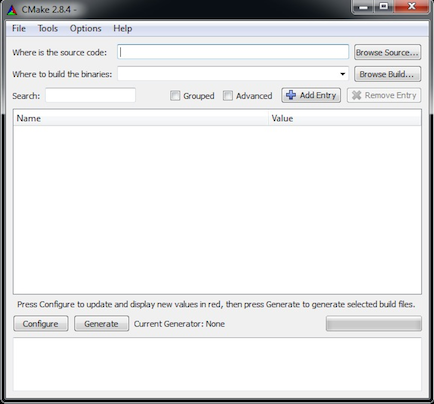
To create configurations suitable for compiling libSBML using the native Windows build tools, we recommend using CMake to generate them. (If you do not already have CMake installed on your system, please begin by downloading a copy of CMake version 2.8.4 or later from [cmake.org](http://www.cmake.org/).) Once you have CMake installed, also download and unzip the libSBML source code archive from the [download area on SourceForge.net](http://sf.net/projects/sbml/files/libsbml/5.2.0) and save the archive somewhere on your file system, then unpack it.

Before going further, an issue on Windows concerns the dependency libraries on which libSBML depends. CMake will try to find all the dependencies for the default options, which leads to problems if they do not exist. We recommend Windows users download the dependencies we have prepackaged at the following download location:

<https://sourceforge.net/projects/sbml/files/libsbml/win-dependencies>

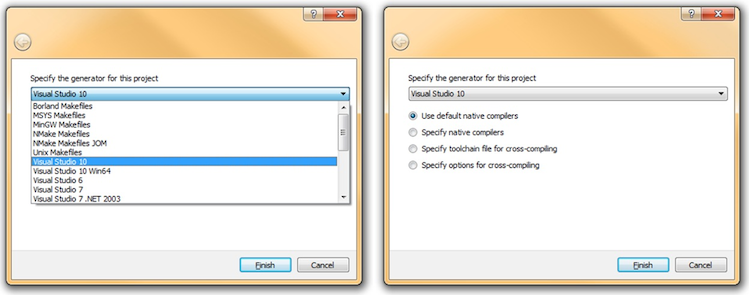
Download and extract this file into the same folder where you unpacked the libSBML source distribution. CMake will look for these dependencies in a folder called **dependencies** directly below the libSBML root folder.

Once you have the libSBML sources and the dependency libraries unpacked on your system, start up the CMake graphical user interface (GUI). It will look something like the following screenshot:



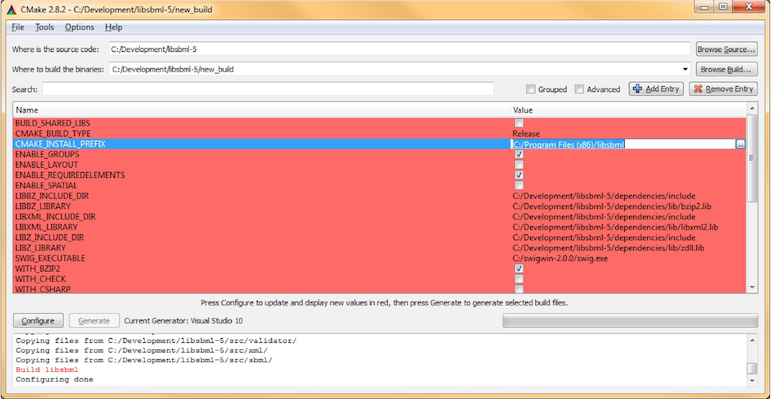
*Screenshot of CMake when it first starts up.*

Click on the **Browse Source...** button and navigate to the directory where you unpacked the libSBML source code archive on your file system. CMake should automatically fill in the next line, **Where to build the binaries**, using a subdirectory named **build** within the directory where your source code is located. (If it does not, fill in the field in the CMake interface yourself.) Then, click on the **Configure** button. CMake should display a dialog similar to the following.



*Screenshot of the****Configure****screen under Microsoft Windows 7. Choose the settings appropriate for the build environment you are using and click the****Finish****button.*

After you close the configuration screen, CMake will populate the options area with various options it reads from the libSBML configuration files. These new options are displayed in red, as illustrated in the following screenshot.



*Screenshot of the CMake interface after the configuration step.*

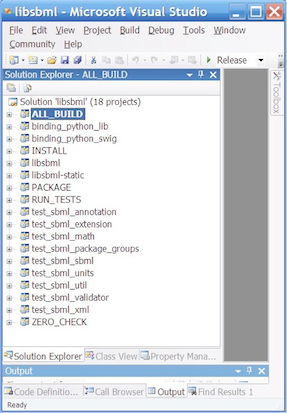
In this panel, you can set values for various libSBML configuration/build options. For example, you can set the location into which the compiled libSBML libraries will be installed by clicking on the line for**CMAKE\_INSTALL\_PREFIX** and replacing the default value with a value of your choosing. To find out the meaning of a particular option, hover your mouse/pointer over the item for a few seconds until the tooltip text appears.

Select the options with which you wish to build libSBML. All the options for configuring libSBML, including language bindings and SBML Level 3 packages, are listed here and may be selected/deselected as required. For instance, to include the Java language bindings, click the check-box for **WITH\_JAVA**. Once you are done configuring, click on CMake's **Configure** button. CMake will process the configuration, and if it lacks any information, it will highlight the options in red. You may need to iterate between setting options and clicking the **Configure** button until there are no options left in red and the **Generate** button becomes enabled.

Next, click on the **Generate** button. This will cause CMake to create project configuration files ("Solutions") for the compilation environment you selected in the beginning (e.g., Visual Studio 2010).

### *3.2 Compiling and installing using MSVC*

The MSVC Solution will contain a number of projects files, depending on the configuration selected. The following screenshot illustrates what you should see when you use the Solution Explorer in MSVC, and some of the most important targets that you will find there:



*Screenshot of MSVC's Solution Explorer when viewing the MSVC Solution files generated using CMake.*

* **ALL\_BUILD**: This target builds all the libSBML project files; that is, all projects except the ones that involve installation, packaging or testing.
* **INSTALL**: This target will install the compiled binaries to the directory specified for the **CMAKE\_INSTALL\_PREFIX** option of the CMake configuration step.
* **ZERO\_CHECK**: This target is merely a verification project; it is used to check whether there have been any changes to the configuration since the last time it was run. It is invoked automatically whenever any other target is being built.
* **PACKAGE**: This target creates binary installers for libSBML. If the Nullsoft scriptable installation system, NSIS ([http://nsis.sf.net](http://nsis.sf.net/)) is available, an installer is created for the current platform. Most users will not need to use this build target/Solution.
* **RUN\_TESTS**: This target can be used to test all the libraries built. It is only available if the **WITH\_CHECK** option is selected during the CMake configuration step. (Note: the [libcheck](http://check.sourceforge.net/" \t "_blank) library upon which this facility depends is incompatible with MSVC 7, so you will need to use a newer version of MSVC if you want to try **RUN\_TESTS**.) The checks will fail if Windows is unable to locate the dynamic libraries, or for each language binding if it is unable to locate both the binding library and the libSBML native library.

Other projects generated by the libSBML CMake configuration system and listed in the MSVC Solution explorer are named to indicate the intended target. Some examples include the following:

* **binding\_python\_lib**: builds the Python language bindings **\_libsbml.pyd** file.
* **binding\_java\_classes**: builds the Java language bindings class JAR file.
* **example\_c\_convertSBML**: builds the **convertSBML** example program in C.
* **example\_java\_addCVTerms**: builds the **addCVTerms** example program in Java.
* **test\_sbml\_math**: builds the tests in the libSBML **src/math** subdirectory.

A typical procedure for building libSBML using MSVC consistes of performing the following steps:

1. Select and run the **ALL\_BUILD** target. After a successful build, MSVC will put the libSBML library files and DLL into the **Release** subdirectory of the location specified using CMake. This includes the language bindings, which will be placed in a subdirectory of the **Release** directory. For example, C# files will appear in **Release/csharp**.
2. Select and run the **INSTALL** target. This will install the compiled libSBML library and associated files into the directory determined by the **CMAKE\_INSTALL\_PREFIX** CMake configuration variable.

Once the libSBML files are installed as described in the sections above, you may need to perform additional steps so that software can find the libSBML library files at run time. Please see the instructions on [*Making libSBML accessible to your software*](http://sbml.org/Software/libSBML/docs/cpp-api/libsbml-accessing.html) provided on a separate page of this documentation.