Building the SheafSystem™ on Windows

# Copyright Notice

© 2016 Limit Point Systems, Inc.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

# Platform

The SheafSystem is supported for Windows 7 and 8 using Visual Studio 2015

# Software Prerequisites

Building the SheafSystem requires the following packages. It is known to build with the versions indicated; it may work with other versions.

* CMake 3.5.0. Cross platform build tool; download from www.cmake.org.
* Visual Studio 2015. C++ compiler; available from Microsoft.
* hdf5 1.8.16, File format and i/o library; download from www.hdfgroup.org.
* doxygen 1.8.9.1. Documentation generator; download from www.doxygen.org
* graphviz 2.38. Graph drawing package used by doxygen; download from www.graphviz.org.
* 7-zip 9.20 or newer. File archiving utility used to extract SheafSystem source from distribution file. Down load from www.7-zip.org.

Binary distributions with Windows installers are available for CMake, hdf5, doxygen, graphviz and 7-zip; just use the default installation procedure.

# Building and installing the SheafSystem

Extract the package in a directory of your choice.

We'll assume you've downloaded the source as a zip file, SheafSystem-<version>.zip. (For instance, the SheafSystem Github page provides a link to download SheafSystem-development.zip). In Windows Explorer, with 7-zip installed, just navigate to the folder containing the download and right-click on the file and select one of the extraction options to extract into a location of your choice. The package will extract into <your choice>/SheafSystem-<version>. From here on we'll refer to that location as <sheaf\_system\_source>

Configure with CMake

Start the CMake application. In the "where is the source code" box enter or browse to <sheaf\_system\_source>. In the "where to build the binaries" box enter <sheaf\_system\_source>/build. Click on the configure button, then click the yes button to create the build directory and click the finish button to accept the default generator (Visual Studio). Configuring will start, it may take a while. Typically, several messages will be displayed about not being able to find some prerequisites.

There are two check boxes in the upper right part of the display. The "Advanced" check box toggles the display between "basic" and "advanced" mode. In basic mode the display shows only the variables you need to set to configure the system. In advanced mode, the display shows a large number of variables detailing the configuration process. Toggle the display to basic mode.

The "Grouped" check box toggles the display between "ungrouped" and "grouped" modes. In "ungrouped" mode the variables are listed in alphabetical order; in "grouped" mode, the variables are organized into an outline-like display. Choose whichever mode you find most appealing, but if in "grouped" mode, expand the headings so that all variables are visible.

There are three groups of variables you need to review, and perhaps set, to configure the SheafSystem: the CMAKE\_ variables, the SHEAFSYSTEM\_ variables, and the PREREQ\_ variables.

Set the CMAKE\_ variables

The CMAKE\_ variables control functionality built into to CMake.

These variables are set by direct entry: select a variable value to begin editing. Variables that are paths can set using a file browser dialog by clicking on the "..." box at the extreme right of the value field.

The command line entry method, available in Linux, is not available in Windows.

There is only one CMAKE\_ variable in Windows:

CMAKE\_INSTALL\_PREFIX (type STRING): the absolute path to the installation destination for the INSTALL target.

Set the SHEAFSYSTEM\_ variables.

The SHEAFSYSTEM\_ variables control options for building the system. The direct entry method applies to these variables.

The SHEAFSYSTEM\_ variables are:

SHEAFSYSTEM\_BUILD\_BINDINGS (type BOOL): if ON, build the Java, C#, and Python bindings for the SheafSystem libraries. See Appendix A for a discussion of building the bindings.

SHEAFSYSTEM\_BUILD\_SHEAFSCOPE (type BOOL): if ON, build the SheafScope Java application for browsing hdf files produced by the SheafSystem. Requires building the bindings.

SHEAFSYSTEM\_DOC\_STATE (type STRING): selects whether to build the developer ("Dev") or User documentation. The User documentation does not include source file listings and documentation for private members.

SHEAFSYSTEM\_ENABLE\_WIN32\_MP (type BOOL): if ON, add the /MP switch to the compile flags to enable multiprocessing during the compile and link.

SHEAFSYSTEM\_INSTALL\_DOCS (type BOOL): if ON, install the documentation when installing the libraries.

Set the PREREQ\_ variables.

The PREREQ\_ variables control the search for the prerequisites. There are two methods for setting these variables: direct entry and environment variable entry. Direct entry is as described above. To use environment variable entry, set an environment variable of the same name to the desired value before invoking CMake. Note that no matter which of these methods is used, it is important to set the value correctly. Incorrect values may produce unpredictable and hard to interpret results. In this case, it is often best to just delete the build directory and try again from Step 2!

For PREREQ\_ variables that describe the path to an executable, a third method is available. If the executable is installed in a location that is in your user PATH or the system Path environment variables, CMake will find it and automatically set the PREREQ\_ variable. When using the PATH method, if a prerequisite is not installed in one of the standard system locations, the prerequisite install location should be prepended, not appended, to the existing PATH variable. Then CMake will find the prerequisite version before it finds some other version which may have been previously installed in one of the standard locations in your system.

When the prerequisites are all found successfully, CMake will write the file set\_prereq\_vars.bat into the build directory. This file is a batch script for setting environment variables for all the PREREQ\_ variables. It can be used on subsequent builds to simplify setting the PREREQ\_ variables - just run the script in a command window before running CMake.

The PREREQ variables are:

PREREQ\_DOXYGEN\_EXECUTABLE (type FILEPATH): the absolute path to the doxygen executable. This prerequisite is optional, but if it is not found, no documentation will be generated.

PREREQ\_GRAPHVIZ\_DOT\_EXECUTABLE (type FILEPATH): the absolute path to the Graphviz dot executable. This prerequisite is optional, but if not found, documentation will not contain diagrams.

PREREQ\_HDF5\_CONFIG\_DIR (type PATH): the absolute path to the directory containing the hdf5-config.cmake file. If HDF5 is installed in the standard location, this will be C:\Program Files\HDF\_Group\HDF5\1.8.16\cmake.

Configure and generate

Click the configure button again. If it completes without error, click the generate button. Otherwise, correct the variables as needed and click configure again. When you've successfully generated, you're done, exit CMake.

Build and install

In the Windows Explorer navigate to <sheaf\_system\_source>/build. Double click on SheafSystem.sln to start Visual Studio. In Visual Studio select Debug\_contracts in the Solution Configurations box. In the Solution Explorer pane, right click on the INSTALL target and select Build. Repeat for the Debug\_no\_contracts and RelWithDebInfo\_no\_contracts configurations.

1. Building bindings for Java, C# and Python

Bindings for Java, C#, and python are currently under development and are not a supported feature. This appendix documents how to build the bindings in the current state of development.

* 1. Software Prerequisites

In addition to the prerequisites listed in section 3, building the bindings requires the following packages. It is known to build with the versions indicated; it may work with other versions.

* Java JDK or OpenJDK version 1.7. Java development platform. Down load from www.oracle.com or openjdk.java.net.
* VTK version 7.0.0. Visualization library; download from www.vtk.org.
* Swig version 3.0.10. Download from www.swig.org
* .NET SDK. CSharp compiler. Available from Microsoft.
* Python version 3.5.2. Programming language. Download from www.python.org.
* JMF (Java media framework) version 2.1.1e. Java framework for video. Download from www.oracle.com.
  1. Building and installing the prerequisites

Except for VTK, use the standard build and/or install procedure each package provides. For Python, select custom install and select the debug features.

* 1. Building and installing vtk

The vtk package requires special handling. There is a bug in this version of the vtk cmake files that doesn't properly initialize the naming suffix for debug libraries, so you must do it manually.

* + 1. Extract the package in a directory of your choice.

In Windows Explorer, navigate to the folder containing the vtk down load package, vtk-7.0.0.tar.gz or vtk-7.0.0.zip, either will work. Right click on the file, select the 7 zip menu item, then select your choice of extraction option to extract the files into a location of your choice. The files will extract into <your choice>/VTK-7.0.0. We'll refer to this location as <vtk\_source> from here on.

* + 1. Remove the build folder

If you've previously built VTK from the source folder, there may be a leftover build folder. Remove it.

* + 1. Configure with CMake

Start the CMake application. In the "where is the source code" box enter or browse to <vtk\_source>. In the "where to build the binaries" box enter <vtk\_source>/build. Click on the configure button, then click the yes button to create the build directory and click the finish button to accept the default generator (Visual Studio). Configuring will start, it may take a while.

When configuring completes, make sure the "advanced" box is checked so all the configuration variables will show.

Click the "add new entry" button and add a variable CMAKE\_DEBUG\_POSTFIX of type string with value "\_d" (quotes are not part of the value).

Make sure the BUILD\_SHARED\_LIBS option is ON

Make sure VTK\_WRAP\_JAVA is ON

Set the CMAKE\_INSTALL\_PREFIX variable to your choice of installation location, for instance <absolute path to install parent>/vtk-7.0.0-shared-debug. (We'll refer to this location as <vtk\_install> below.) VTK does not support multi-configuration installations, so choose a different install location for each configuration.

You can avoid some annoying but harmless warnings later (when linking to the vtk libraries while building SheafSystem) by replacing "/Zi" with "/Z7" in the variables CMAKE\_C\_FLAGS\_DEBUG and CMAKE\_CXX\_FLAGS\_DEBUG.

Finally, you can make the build go much faster by setting CMAKE\_CXX\_MP\_FLAG to ON and setting CMAKE\_CXX\_MP\_NUM\_PROCESSORS to an appropriate value for your workstation.

Click the configure button again. When it completes, click the generate button. When generating completes, you're done, exit CMake.

* + 1. Build and install the shared-debug configuration

In the Windows Explorer navigate to <vtk\_source>/build. Double click onVTK.sln to start Visual Studio. In Visual Studio select Debug in the Solution Configurations box. In the Solution Explorer pane, right click on the "INSTALL" target and select Build. Visual Studio will build and install the shared-debug configuration of the VTK libraries.

* + 1. Build and install the shared-release configuration

Repeat steps 2 through 4, setting CMAKE\_INSTALL\_PREFIX to a different location, for instance VTK\_7.0.0-shared-release. Choose the Release configuration in Visual Studio and build the install target.

* + 1. Build and install the shared-relwithdebinfo configuration

Repeat steps 2 through 4, setting CMAKE\_INSTALL\_PREFIX to a different location, for instance VTK\_7.0.0-shared-relwithdebinfo. Choose the RelWithDebInfo configuration in Visual Studio and build the install target.

* + 1. Build and install the static libraries (optional)

Repeat steps 2 through 4 for the static-debug and static-release configurations. Set BUILD\_SHARED\_LIBS and VTKWRAP\_JAVA to OFF and choose appropriate installation locations.

* 1. Building and installing the bindings

Follow the steps for building and installing the SheafSystem above, but with the following modifications:

**Step 4': Set the SHEAFSYSTEM\_ variables**

Set the SHEAFSYSTEM\_BUILD\_BINDINGS variable to ON. If desired, also set the SHEAFSYSTEM\_BUILD\_SHEAFSCOPE variable to ON. Then click configure to run the configuration process. Typically, CMake will display several messages about not being able to find the additional prerequisites required by the bindings.

**Step 5': Set the PREREQ\_ variables.**

When the SHEAFSYSTEM\_BUILD\_BINDINGS option is selected, a number of additional PREREQ\_ variables appear. The methods described in Step 5 for setting the original PREREQ\_ variables also apply to these additional variables.

The additional PREREQ\_ variables are:

PREREQ\_JAVA\_HOME (type PATH): the absolute path to the top level of the Java installation.

PREREQ\_JMF\_EXECUTABLE (type FILEPATH): the absolute path to the jmfinit executable.

PREREQ\_PYTHON\_EXECUTABLE (type FILEPATH): the absolute path to the python interpreter executable.

PREREQ\_SWIG\_EXECUTABLE (type FILEPATH): the absolute path to the swig executable.

PREREQ\_VTK\_CONFIG\_DIR (type PATH): the absolute path to the directory containing VTKConfig.cmake. If built as described above, this should be <vtk\_install\_location>/lib/vtk-7.0. Use the shared-debug, shared-release, or shared-relwithdebinfo installation, depending on which configuration of the SheafSystem you are building.