Building the SheafSystem™ on Linux

# 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 the Redhat Enterprise Linux platform and it's variants, CentOS and Scientific Linux. Fedora should also work. The instructions below assume you are running one of these distributions.

# 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; available from various Linux repositories or down load from www.cmake.org.
* g++ 4.9.3. Gnu C/C++ compiler; available from various Linux repositories or down load from gcc.gnu.org.
* hdf5 1.8.16. File format and i/o library; down load from www.hdfgroup.org.
* doxygen 1.8.5. Documentation generator; available from various Linux repositories or down load from www.doxygen.org. (Any subsequent release will work, but all have a bug in the generated class list, see bug 764711 at bugzilla.gnome.org.)
* graphviz 2.38.0. Graph drawing package used by doxygen; available from various Linux repositories or down load from www.graphviz.org.
* 7-zip 15.09. File archiving utility used to extract SheafSystem source from distribution file; available from various Linux repositories or down load from www.7-zip.org.

## Building and installing the prerequisites

For all the prerequisites except hdf5, use the standard build and/or install procedure the package provides.

## Building and installing hdf5

Binary distributions of hdf5 are available for various Linux platforms, but they do not include the CMake configuration files required by the SheafSystem. Click on the "How to build HDF5 with CMake using a platform configuration file" link on the HDF5 release page and follow the instructions there to down load the CMake version of the source and build hdf5 with CMake. The build produces a binary package HDF5-1.8.16-Linux.tar.gz. Untar this package in an installation location of your choice.

# Building and installing the SheafSystem

* 1. 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). Extract the source into a location of your choice.

* cd <your choice again>

7za x <path to>/SheafSystemTest-<version>.zip

The package will extract into <your choice again>/SheafSystem-<version>. From here on we'll refer to that location as <sheaf\_system\_source>

* 1. Start a new shell

We don't want any old values for environment variables hanging around. In particular, we don't want those that were set by sourcing the set\_env\_vars script in a previous build attempt, see below. So start a new shell.

* 1. Cd to the SheafSystem directory.
* cd <sheaf\_system\_source>
  1. Remove the build directory, if any

CMake caches results of previous build attempts both in memory and on disk in the build directory. Starting with old values can sometimes produce hard to explain results.

* rm -rf build
  1. Create a new build directory
* mkdir build
  1. Cd to the build directory
* cd <sheaf\_system\_source>/build
  1. Set the compiler environment variables

The only reliable way to get the right compilers is to set the environment variables CC and CXX to the absolute paths to the desired C and C++ compilers, respectively. This is particularly important if the compilers required by the SheafSystem are not the default compilers installed in your operating system. These variables must be set before running CMake, the compiler cannot be changed from within CMake.

* 1. Run the CMake GUI

Make sure the CMake executables cmake and ccmake are in your path, then type:

* ccmake ..

The CMake GUI will start. Type "c" to configure. Typically, it will then display several messages about not being able to find the prerequisites. Type "e" to exit this message display and ccmake will display a table of CMake variables and their values. As you move the cursor through the rows of the table, a short description of each variable will appear in the status line near the bottom of the display.

Typing "t" 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. 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.

* 1. Set the CMAKE\_ variables

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

There are two methods for setting these variables, direct entry and command line entry. To use direct entry, move the cursor to select a variable and type <enter> to begin editing the value. Type <enter> again to end editing. Path completion using the <tab> key is supported for variables that are paths.

To use command line entry type -D<variable name>:<type>=<variable value> on the command line when invoking ccmake. If the type specifier is omitted or incorrect, the resulting behavior can be hard to interpret. Also, values specified on the command line cannot be changed interactively.

The CMAKE\_ variables are:

CMAKE\_BUILD\_TYPE (type STRING): The kind of code that will be generated by the build. Allowed values are "Debug\_contracts", "Debug\_no\_contracts", and "Release\_no\_contracts". Debug\_contracts sets the -g compiler option to produce debugging information and enables contracts (assertions), Debug\_no\_contracts sets the -g option but disables contracts, and Release\_no\_contracts enables optimization and disables contracts. To view the precise compiler options that these build types imply, toggle to advanced mode and view the variables CMAKE\_CXX\_FLAGS\_<build\_type>. The contracts are very useful for debugging but can make the system run much slower than without contracts.

CMAKE\_INSTALL\_PREFIX (type PATH). The absolute path to the installation destination for the make install target.

* 1. Set the SHEAFSYSTEM\_ variables.

The SHEAFSYSTEM\_ variables control options for building the system. The direct entry and command line entry methods apply 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\_INSTALL\_DOCS (type BOOL): if ON, install the documentation when installing the libraries.

* 1. Set the PREREQ\_ variables.

The PREREQ\_ variables control the search for the prerequisites. There are three methods for setting these variables: direct entry, command line entry, and environment variable entry. Direct entry and command line entry are as described above. To use environment variable entry, set an environment variable of the same name to the desired value before invoking ccmake. 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 5!

For PREREQ\_ variables that describe the path to an executable, a fourth method is available. If the executable is installed in a location that is in your PATH environment variable, ccmake 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 files set\_prereq\_vars.csh and set\_prereq\_vars.sh into the build directory. These files are C-shell and bash scripts, respectively, for setting environment variables for all the PREREQ\_ variables. They can be used on subsequent builds to simplify setting the PREREQ\_ variables - just source the appropriate script before running cmake or ccmake.

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 as described above, this path will be HDF5-1.8.16-Linux/HDF\_Group/HDF5/1.8.16/share/cmake.

* 1. Configure and generate the build system.

After entering all the configuration variables, type "c" to run the configuration process again. Various status messages will be displayed in the lower part of the window. If any prerequisites cannot be found, error messages will be displayed. Type "e" to exit the error message display and adjust the PREREQ\_ variables as needed. Once the PREREQ\_ variables have all been set correctly, the variable table will update, showing any new results in red. Type "c" again and the "g" (generate) option will be enabled. Type "g" to generate the make files for the build system and exit.

* 1. Source the set\_env\_vars script

In bash, source the set\_env\_vars.sh script.

* source set\_env\_vars.sh

In csh or tcsh, source the set\_env\_vars.csh script

* source set\_env\_vars.csh
  1. Execute make

While still in the <sheaf\_system\_source>/build directory:

* make

will build docs, shared libs, and static libs.

* make lib

will build only the shared libraries.

* make doc

will make the reference documentation.

* make install

will install the docs, libraries, and tools in CMAKE\_INSTALL\_PREFIX.

The -j <number of jobs> option to make, which builds in parallel, is very useful for the default, lib, and install targets. Doesn't help with the doc target.

* make help

will display more about build system capabilities.

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. Available from various Linux repositories or down load from www.oracle.com or openjdk.java.net.
* VTK version 7.0.0. Visualization library; download from www.vtk.org.
* Swig version 1.3.40. Available from various Linux repositories or download from www.swig.org
* Mono version 2.10.8. C# compiler for Linux. Available from various Linux repositories (package mono-core is enough) or download from www.mono-project.org.
* Python version 2.6.6. Programming language. Available from various Linux repositories or 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.

* 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.

Navigate to the folder containing the vtk down load package, vtk-7.0.0.tar.gz. Untar 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. Cd to the vtk\_source directory.
* cd <vtk\_source>
  + 1. Remove the build directory, if any

CMake caches results of previous build attempts both in memory and on disk in the build directory. Starting with old values can sometimes produce hard to explain results.

* rm -rf build
  + 1. Create a new build directory
* mkdir build
  + 1. Cd to the build directory
* cd <vtk\_source>/build
  + 1. Set the compiler environment variables

The only reliable way to get the right compilers is to set the environment variables CC and CXX to the absolute paths to the desired C and C++ compilers, respectively. This is particularly important if the compilers are not the default compilers installed in your operating system. These variables must be set before running CMake, the compiler cannot be changed from within CMake. You may also need to set LD\_LIBRARY\_PATH to include paths to the compiler specific libraries, for instance <compiler installation>/lib64.

* + 1. Run the CMake GUI and define CMAKE\_DEBUG\_POSTFIX

The naming suffix for debug libraries is set by defining the variable CMAKE\_DEBUG\_POSTFIX. The only way this variable can be set with the Linux version of CMake is to define it on the command line when invoking ccmake.

Make sure the CMake executables cmake and ccmake are in your path, then type:

* ccmake .. -DCMAKE\_DEBUG\_PREFIX:STRING=\_debug
  + 1. Set the other CMAKE variables

Type "c" to begin configuration. Configuring will start, it may take a while.

Make sure the BUILD\_SHARED\_LIBS option is ON

Set CMAKE\_BUILD\_TYPE to Debug.

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.

Make sure VTK\_WRAP\_JAVA is ON

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

To build and install the debug configuration with shared libraries:

* make -j<num> install
  + 1. Build and install the shared-release configuration

Repeat steps 3 through 10, setting CMAKE\_BUILD\_TYPE to Release. Set CMAKE\_INSTALL\_PREFIX to a different location, for instance VTK-7.0.0-shared-release. Configure, generate, and make install.

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

Repeat steps 3 through 10. Set BUILD\_SHARED\_LIBS and VTKWRAP\_JAVA to OFF, set CMAKE\_BUILD\_TYPE to Debug, and choose an appropriate installation location. Make sure that CMAKE\_C\_FLAGS AND CMAKE\_CXX\_FLAGS contain the -fPIC option. Configure, generate, and make install. Repeat for build type Release.

* 1. Building and installing the bindings

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

**Step 10': Set the SHEAFSYSTEM\_ variables**

Set the SHEAFSYSTEM\_BUILD\_BINDINGS variable to ON. If desired, also set the SHEAFSYSTEM\_BUILD\_SHEAFSCOPE variable to ON. Then type "c" to run the configuration process. Typically, ccmake will display several messages about not being able to find the additional prerequisites required by the bindings. Type "e" to exit this message display and ccmake will display the table of variables.

**Step 11': Set the PREREQ\_ variables.**

When the SHEAFSYSTEM\_BUILD\_BINDINGS option is selected, a number of additional PREREQ\_ variables appear. The methods described in Step 10 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\_MCS\_EXECUTABLE (type FILEPATH): the absolute path to the mcs executable in the mono package.

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/cmake/vtk-7.0. Choose the shared-debug installation if building the Debug\_contracts or Debug\_no\_contracts configuration. Choose the shared-release installation if building the Release\_contracts or Release\_no\_contracts configuration.