Building SheafSystemTest™ 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

SheafSystemTest, the test suite for the SheafSystem™ libraries, 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 SheafSystemTest 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. The compiler version should be the same as the SheafSystem libraries were built with.
* 7-zip 15.09. File archiving utility; available from various Linux repositories or down load from www.7-zip.org.
* SheafSystem, installed libraries or development tree.

Use the standard build and/or install procedure each package provides.

# Building and running SheafSystemTest

1. Extract the source in a location of your choice

We'll assume you've down loaded the source as a zip file, SheafSystemTest-<version>.zip. (For instance, the SheafSystemTest Github page provides a link to download SheafSystem-master.zip). Extract the source into a location of your choice.

* cd <your choice>

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

The files are extracted into <your choice>/SheafSystemTest-<version>. We'll refer to this directory as <sheaf\_system\_test\_source> from here on.

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.

Cd to the SheafSystemTest directory.

* cd <sheaf\_system\_test\_source>

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

Create a new build directory

* mkdir build

Cd to the build directory

* cd <sheaf\_system\_test\_source>/build

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.

Run the CMake GUI

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

* ccmake ..

The CMake GUI will start. Typically, it will 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.

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.

There is only one CMAKE\_ variable:

CMAKE\_BUILD\_TYPE (type STRING): the kind of code that will be generated by the build. Allowed values are "Debug\_contracts" and "Release\_no\_contracts". Debug\_contracts sets the -g compiler option to produce debugging information and enables contracts (assertions). 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.

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.

There is only one SHEAFSYSTEM\_ variable:

SHEAFSYSTEM\_TEST\_JOBS (type STRING): the number of test jobs CTest will execute concurrently. Concurrent tests make the testing take less elapsed time but can make the results harder to read.

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!

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.

There is one PREREQ variable:

PREREQ\_SHEAFSYSTEM\_HOME (type PATH): the absolute path to the top level directory of the SheafSystem installation or to the build directory of a SheafSystem development tree.

Configure and generate the build system.

After entering all the configuration variables, type "c" to run the configuration process. 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.

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

Execute make

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

* make check

will build and execute the test suite for the selected configuration. The test suite contains several hundred test executables. The Debug\_contracts tests take quite a while; Release\_no\_contracts will run much faster. The -j <number of jobs> option to make, which builds the test executables in parallel, speeds things up.