Building SheafSystemProgrammersGuide™ on Linux

1 Copyright Notice

2 Platform

SheafSystemProgrammersGuide, the tutorial 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.

3 Software Prerequisites

Building SheafSystemProgrammersGuide 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.

4 Building and running SheafSystemProgrammersGuide

Step 1: Extract the source in a location of your choice

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

```
> cd <your choice>
> 7za x <path to>/SheafSystemProgrammersGuide-<version>.zip
```

The files are extracted into <your choice>/SheafSystemProgrammersGuide-<version>. We'll refer to this directory as <prgrmrs_guide_source> from here on.

Step 2: 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.

Step 3: Cd to the SheafSystemProgrammersGuide examples directory.

```
> cd <prgrmrs_guide_source>/examples
```

Step 4: 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
```

Step 5: Create a new build directory

```
> mkdir build
```

Step 6: Cd to the build directory

```
> cd <prgrmrs_guide_source>/examples/build
```

Step 7: 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.

Step 8: 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 a message 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 ia only one group of variables you need to review, and perhaps set, to configure the SheafSystemProgrammersGuide: the PREREQ_ variables.

Step 9: 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_CONFIG_DIR (type PATH): the absolute path to the directory containing the file SheafSystemConfig.cmake. For instance, <installation tree root>/cmake or <source tree root>/build/cmake.

Step 10: Configure and generate the build system.

After entering the configuration variable, 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_ variable as needed. Once the PREREQ_ variable has 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.

Step 11: Execute make

While still in the <prgrmrs_guide_source>/build directory:

```
> make
```

will build the executables for all the examples.

Step 12: Execute the examples

Navigate to the directory containing the executables:

```
> cd <prgrmrs_guide_source>/examples/build/Debug_contracts/bin
```

Some of the examples depend on the output of preceding examples, so you have to execute the examples in order. To execute an example, just type its name, for instance:

```
> example01
```

The CMake build system sets the RPATH property for the execuables, so they should run with setting LD_LIBRARY_PATH.