AlloSystem Install Tutorial

Lance Putnam and Lars Knudsen

May 11, 2020

Contents

1 Introduction

This tutorial explains how to install AlloSystem under Linux Ubuntu, Mac OS X, and Windows. The simplest way of working with AlloSystem is through the command-line interface with tools like `GCC` and `make`. The first part of the tutorial explains how to install the necessary development tools for compiling code. The second part describes how to install AlloSystem. The last part explains how to compile and run custom applications using AlloSystem.

Read the instructions in each section very carefully since skipping an installation step which likely cause things to fail during later steps. During each installation step, be sure to carefully monitor any output for errors. If an error does arise, you can refer to Troubleshooting to find solutions to common problems. In addition, be careful when copy and pasting commands from this document as it can introduce extra spaces leading to malformed paths, etc.

1.1 Supported Operating Systems

Currently, the following operating systems are known to work with AlloSystem:

- Linux Ubuntu
- Mac OS X 10.5 (Leopard) / 10.6 (Snow Leopard) / 10.7 (Lion) / 10.8 (Mountain Lion) / 10.9 (Mavericks) / 10.10 (Yosemite)
- Windows 7 / 8 / 10

If you choose to use operating systems or tools other than those recommended in this tutorial, then *you are on your own*.

2 Get Development Tools

The first step is to obtain the development tools necessary to compile and run AlloSystem programs. It is imperative that you maintain a good internet connection throughout the installation process as many packages need to be downloaded. Skip to the platform you are using and follow the instructions there.

2.1 Linux Ubuntu

Ensure that both `make` and g++ are installed by entering the following command in the terminal:

```
$ sudo apt-get install make g++
```

When that finishes, proceed to Install AlloSystem.

2.2 Mac OS X

2.2.1 Before Beginning

Before beginning to install the development tools, it is crucial that you complete any OS updates. If you don't, the installation of Xcode may fail with an "unknown error". (This is a known issue with OS X 10.6.x Snow Leopard and Xcode 3.2.6.) Do system updates by clicking the Apple symbol in the upper-left corner of the screen and then selecting "App Store..." or "Software Update...". Let the updater finish its course which may require restarting. Repeat this update process until there are no more updates.

2.2.2 Install Development Tools

Open the App Store application and search for "xcode". On the result screen, select and install the Xcode application:

After Xcode installation completes, run Xcode (agreeing to the Xcode license agreement). The last step is to install the command-line tools. If you have OSX 10.9 or above, open the Terminal application, and execute the command

```
xcode-select --install
```

If you have OSX 10.8 or below, you must install the command-line tools from inside of Xcode. This is accessible from the menu item Xcode -> Preferences... -> Downloads.

2.2.3 Install Package Manager

MacPorts is a package management tool for OS X which can be used to install AlloSystem dependencies. Download and install the latest version from http://www.macports.org/install.php. Make sure to install the correct version for your version of OSX (Snow Leopard, Lion, Mountain Lion, Mavericks, etc.).

2.3 Windows and MinGW/MSYS

2.3.1 Install Code Editor

You will need to install an editor that recognizes and syntax colorizes source code. We recommend either Visual Studio Express or Notepad++. Notepad++ is a lightweight text editor and starts up instantaneously (like most modern software should).

2.3.2 Install MinGW-w64 and Msys2

Before beginning, it is important that your Windows user name does not contain any spaces. Msys2 uses this as your user name and if it has spaces in it, it treats them like command delimiters which creates many problems.

Download Mingw-w64/Msys2 and follow the installations instructions at https://www.msys2.org. Make sure to update the package database using `pacman` as instructed.

2.3.3 Start Mingw-w64 Shell

From now on, you will use the Mingw-w64 shell (not Msys2). Double-click `mingw64.exe` in `c:\msys64\` and proceed to 3.

2.3.4 Mingw/Msys Shell Tips

Windows drives, like `C:` and `D:`, can be accessed as `/c` or `/d`, respectively. Use the `mount` command to get an overview of all the drive mappings.

Normally, when you install libraries in a Unix environment, the libraries get placed in `/usr/local/`. Under the Mingw-w64 shell, libraries are instead installed to `/mingw64/`.

Your user directory lives in `c:\msys64\home\`. If you would like to run some commands when the shell starts, then edit the file `.bash_profile` in your user directory. For example, it is usually helpful to add the line `cd /c/code/AlloSystem/` (or wherever you have AlloSystem installed) to `.bash_profile`.

3 Install AlloSystem

At this point, you should have the appropriate development tools installed for your platform. In the following instructions, when we say 'shell' we mean the command-line interface which is MSYS for Windows users and the Terminal application for Mac OS X and Linux users.

First, create a new directory `~/code/`. This can be done from the shell with the command

```
$ mkdir ~/code
```

Next, download the AlloSystem source code from

https://github.com/AlloSphere-Research-Group/AlloSystem/archive/makefile-build.zip.

Within the `.zip` file should be another directory named something like `AlloSystem-makefile-build`. You should move this directory to the directory `~/code/` you just made. After the directory `AlloSystem-makefile-build` is moved, rename it to `allosystem`.

From the shell, change into the AlloSystem directory by executing

```
$ cd ~/code/allosystem
```

If you do '`ls`' you should see something like:

```
$ ls
Makefile                alloGLV         allovsr
Makefile.buildandrun    allocore        doc
Makefile.common         allocv          linux
Makefile.rules          allonect        osx
README.md               alloutil        work
```

Next, you need to install the necessary dependencies for the AlloCore module of AlloSystem. (Windows users only: you must *deactivate all virus checkers and antispyware programs, such as Windows Defender*. These monitoring programs are know to terminate or halt important configuration processes that take place in the following steps. If you do not disable them, then you may experience lockups and/or the libraries will not get installed properly. See also 7.7.) Install dependencies with the following command:

```
$ ./install_dependencies.sh
```

If prompted for your password, enter the same password you use to log in to the OS. Installing all the dependencies may take some time, so be patient. After dependencies are installed, the next step is to build the AlloCore module with the following command:

```
$ make allocore
```

To check if everything installed successfully, build and run an example using the command

```
$ ./run.sh allocore/examples/io/appSimple.cpp
```

If you see a window with animated graphics, then AlloSystem is installed correctly.

4 Build-and-Run Applications

While AlloSystem is intended to be used mainly as a library, it does provide a simple means of building and running custom applications that link to AlloSystem. This is accomplished through the `run.sh` shell script. You can build and run any source (`.cpp`) files having a `main()` function using a command with the format

```
$ ./run.sh path/to/file/myfile.cpp
```

The executable binary is placed in `build/bin/` and has the same name as the source file. If you would only like to build the application, without also running it, then you can append `AUTORUN=0` to the line above, i.e.,

```
$ ./run.sh path/to/file/myfile.cpp AUTORUN=0
```

5 GUI and Sound

The following instructions are only required if you would like to use graphical user interface (GUI) or sound synthesis components in conjunction with AlloSystem. We use GLV as the GUI toolkit and Gamma as the sound synthesis library. Download the source code of each at https://github.com/AlloSphere-Research-Group/GLV/archive/master.zip and https://github.com/LancePutnam/Gamma/archive/master.zip. Unzip them within the directory `~/code` (alongside where you installed AlloSystem). Rename the GLV and Gamma directories to `GLV` and `Gamma`. From the shell, this will look like

```
$ cd ~/code/
$ ls
allosystem
Gamma
GLV
```

Next, build GLV and Gamma using the following commands:

```
$ cd ~/code/allosystem
$ make Gamma
$ make GLV
$ make alloGLV
```

6 Advanced Configuration

6.1 Custom Project Directories

It is possible to configure AlloSystem to use your own custom project directory for its build-and-run facility. We will call this your *user directory* and refer to it in this document generically as <userdir> from this point onward. The actual name of the user directory can be anything you choose as long as it is a valid path, e.g., `myprojects`, `mycode` or your initials, such as `ljp`. Start by creating the directory `~/code/allosystem/<userdir>/`. Next, make a copy of the file `Makefile.usertemplate` and name it `Makefile.user`. `Makefile.user` must be in the folder `~/code/allosystem/` along with `Makefile.usertemplate`. You can do this from the terminal with the commands

```
$ cd ~/code/allosystem/
$ cp Makefile.usertemplate Makefile.user
```

Now, open `Makefile.user` in an editor and change the line reading

```
#RUN_DIRS += foo/
```

to

```
RUN_DIRS += <userdir>/
```

using the same directory name you created in the previous step. Notice the # (comment symbol) is removed. Also, do not forget the trailing slash after <userdir>. Relative paths in `RUN_DIRS` are relative to the the `allosystem/` directory. At this point, AlloSystem will recognize your user directory as a valid build-and-run directory. You can add as many user directories as you like to `RUN_DIRS` remembering to separate them with spaces.

While it is possible to have as many user build-and-run directories, sometimes it is convenient to be able to build and run sources from other directories without always having to add them to the `RUN_DIRS` variable. This can be accomplished by creating a symbolic link in one of your valid build-and-run directories:

```
$ cd ~/code/allosystem/
$ ln -s sourcedir/ <userdir>/targetdir
```

This creates a symbolic link to `sourcedir/` that is treated like the directory <userdir>`/targetdir/`.

6.2 Linking to Other Libraries

If your projects need to link to libraries that are not already used by AlloSystem, then there is some additional configuration required. You can configure this in one of two ways—globally, where all projects link to specified libraries, or locally, where only a specific project links to specified libraries. To set up global linking, simply add the linker flags to your existing `Makefile.user` file. For example, to link to the FFTW and Magick++ libraries that were installed into `/usr/local/`, we would set up `Makefile.user` as follows:

```
# Somewhere in Makefile.user ...

# Append directories to be searched for library include files
CPPFLAGS += -I/usr/local/include/

# Append directories to be searched for library object files
LDFLAGS += -L/usr/local/lib/
```

```
# Append libraries to be linked to
LDFLAGS += -lfftw3f -lMagick++
```

Here, `CPPFLAGS` is a string of C/C++ preprocessor flags and `LDFLAGS` is a string of linker flags. The flag `-I` adds a search path for the `#include` preprocessor directive used when compiling code. The flag `-L` adds a library search path for the linker, and `-l` (lowercase letter l) tells the linker which libraries to link to (omitting the starting `lib` and ending extension from the library object file name).

If you would only like local linking, that is, linking to libraries for only one project, then place all the project's source files along with a file `flags.txt` in a subdirectory of one of your user directories. Then, in `flags.txt`, add all the necessary linker flags in the same way as done above with `Makefile.user`.

6.3 Compiling Your Own Object Code

There comes a time when you would like many separate projects to reuse your own custom code, akin to using a library. For example, you may have several utility functions or classes that you would like all projects to have access to. Let's say you have the two files `myutils.h` and `myutils.cpp` that contain your custom functions and classes. At this point, all that is necessary is to place these files in some directory, say `~/code/allosystem/myutils/`, and add the following line to your existing Makefile.user file:

```
RUN_SRC_DIRS += myutils/
```

7 Troubleshooting

7.1 CasE SensitiviTy

If your commands are not working, it is possible that the case of one or more characters in the command is wrong. Many Unix systems are case sensitive.

7.2 I Can't Find The '~' Directory!

The tilde character '~' designates the home directory of the current user. This is usually where you start out when you first open a terminal. The exact location of ~ varies by platform. If you want to know exactly where it is, enter the following in the terminal

```
$ file ~
```

7.3 'No such file or directory'

This error message is typically the result of misspelling a file or directory name or being in the wrong directory. Note many command-line interfaces are also case-sensitive. Always use tab autocompletion when typing file/directory names to ensure that they exist and are spelled correctly.

7.4 make: Nothing to be done for 'xxx'.

This is related to the target you pass into make. The message can mean either: (1) all dependencies of the target you specified are up-to-date or (2) you specified a file name as a target and there is no rule in the Makefile to handle the target.

7.5 make: *** No rule to make target 'xxx'. Stop.

The Makefile does not contain the rule you specified. The rule is the thing you are specifying after typing `make`. If you are attempting to build and run a source file, then this message means that the directory of the source file does not exist where you specified. A common cause is that you are in the wrong directory. Always use tab autocompletion to ensure that you have file paths correct. Another cause is that you are trying to build and run a source file that is not in `work/` or one of your build-and-run user directories (6.1).

7.6 Bash Script (.sh file) Does Not Execute (Properly)

If you have problems trying to execute a Bash script (`.sh` file), then it is possible that its file mode is incorrect. Make the script executable by running

```
$ chmod u+x <filename>.sh
```

7.7 MinGW/MSYS: Freezes When Running `configure` or `make` Commands

It's possible another application (e.g., anti-virus or anti-spyware) is interfering with MinGW's emulation of the Unix fork command. See this related Cygwin issue: http://cygwin.com/faq/faq.html#faq.using.bloda.

7.8 MinGW/MSYS: Spaces in User Name

MSYS simply does not like user names with spaces in them. If your user name contains spaces, then it will create problems with many of the build tools. To solve the problem, you must change your user name in MSYS. Add to the beginning of `C:\MinGW\msys\1.0\msys.bat` a line having the format

```
set USERNAME=<myname>
```

where `<myname>` is replaced with a new user name, such as your initials, all in lower-case that does not contain spaces or other punctuation. Open a new terminal by running `msys.bat` for the changes to take effect.

7.9 MinGW/MSYS: Permission Denied

If you see a message 'Permission denied', it is possible that the Windows Application Experience service is not configured properly. Usually, logging out of Windows and logging back in fixes the problem (no need to restart Windows). If the problem persists, you will need to manually configure Windows group policies.

To configure group policies, click the Start button and in the search bar type: `gpedit.msc`. Click this file to start the Local Group Policy Editor. Next, navigate to `Local Computer Policy -> Computer Configuration -> Administrative Templates -> Windows Components -> Application Compatibility` and do the following

- Select Turn Off Application Compatibility Engine
- Select Enabled under the Settings tab
- Select Turn off Program Compatibility Assistant
- Select Enabled under the Settings tab

See http://www.blackviper.com/windows-services/application-experience/ for more information.

7.10 MinGW/MSYS: '_hypot' was not declared in this scope

If you get this error when compiling, open the file `C:/MinGW/include/math.h`, go to the line number indicated in the error message and change the function name `_hypot` to `hypot`.

7.11 OS X: Problems After Upgrading OS

The most common issue after upgrading OS X is that the standard library headers (e.g., `math.h`, `stdio.h`) are not found when compiling code. Here, the problem can likely be traced to an incorrect system path in `Makefile.common`. You will want to ensure that the compiler flag `-isysroot` is set to the latest SDK found in `/Applications/XCode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/`.

One should also ensure that the latest version of Xcode is installed from the App Store and, afterwards, that the latest command-line tools are installed by executing the following from the command-line

```
$ xcode-select --install
```

If you are using the MacPorts package manager, you will also need to reinstall the latest version of MacPorts for your OS version.

7.12 OS X: '<cmath> not found' When Compiling Sources

This occurs for users with OSX 10.7 and above who have installed Xcode 4.4.1 and above. The cause is that the standard libraries packaged with Xcode are not found by the build system. To fix the problem, ensure that Xcode.app is placed in `/Applications` and not `~/Applications` or some other folder. If you cannot navigate to `/Applications` from Finder, then open a Terminal and type

```
$ open /Applications
```

7.13 OS X: 'Agreeing to the Xcode license requires admin privileges, ...'

After upgrading Xcode, you may get a message in the terminal such as the following: 'Agreeing to the Xcode license requires admin privileges, please re-run as root via sudo.' You can fix this by running the command that triggered the message with `sudo` in front. So if the command you tried was `$ make work/mycode.cpp` try instead `$ sudo make work/mycode.cpp`. Alternatively, you can start Xcode and agree to the license when it asks. After agreeing, you can quit Xcode.

7.14 OS X: Problems With Multiple Versions of Xcode

If you have older versions of Xcode installed and want to remove them (i.e., to save space or fix installation problems) you can uninstall them by running in a terminal

```
$ sudo /Developer/Library/uninstall-devtools --mode=all
```

Typically, you will want to remove old versions *before* trying to install a newer version. If you want to keep multiple versions of Xcode installed on your machine, you can switch between them by running

```
$ sudo xcode-select -switch [path to Xcode]
```

where the path after `-switch` is the version of Xcode you want to use, e.g. `/Applications/Xcode.app/`.