AlloSystem Install Tutorial

Lance Putnam and Lars Knudsen

February 5, 2014

Contents

1 Introduction

This tutorial explains how to install AlloSystem under Linux Ubuntu, Mac OS X, and Windows. The main means of operating AlloSystem is through a Unix-like shell and GNU tools GCC and make. The first part of the tutorial explains how to set up a Unix environment and install the necessary tools. The second part describes how to install AlloSystem. The last part explains how to compile and run custom applications using AlloSystem.

1.1 Supported Operating Systems

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

- Linux Ubuntu

- Mac OS X 10.5.x (Leopard) / 10.6.x (Snow Leopard) / 10.7.x (Lion) / 10.8.x (Mountain Lion) / 10.9.x (Mavericks)

- Windows 7 / 8

For Windows users, Linux Ubuntu can be installed easily and safely using Wubi. 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. 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 3.

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 "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

We will use MacPorts, a package management tool for OS X, 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++. Visual Studio is a full-blown IDE while Notepad++ is a lightweight text editor. Notepad++ is preferred as it starts up much more quickly.

2.3.2 Install MinGW and MSYS

The simplest way to install MinGW/MSYS is to use the automated GUI installer. Download this from
http://sourceforge.net/projects/mingw/files/latest/download
After the download completes, run the installer. When the MinGW Installation Manager (or mingw-get GUI) window pops up (Fig. 1), select 'Basic Setup' in the left pane and then select the following packages to 'Mark for Installation':

- mingw32-base

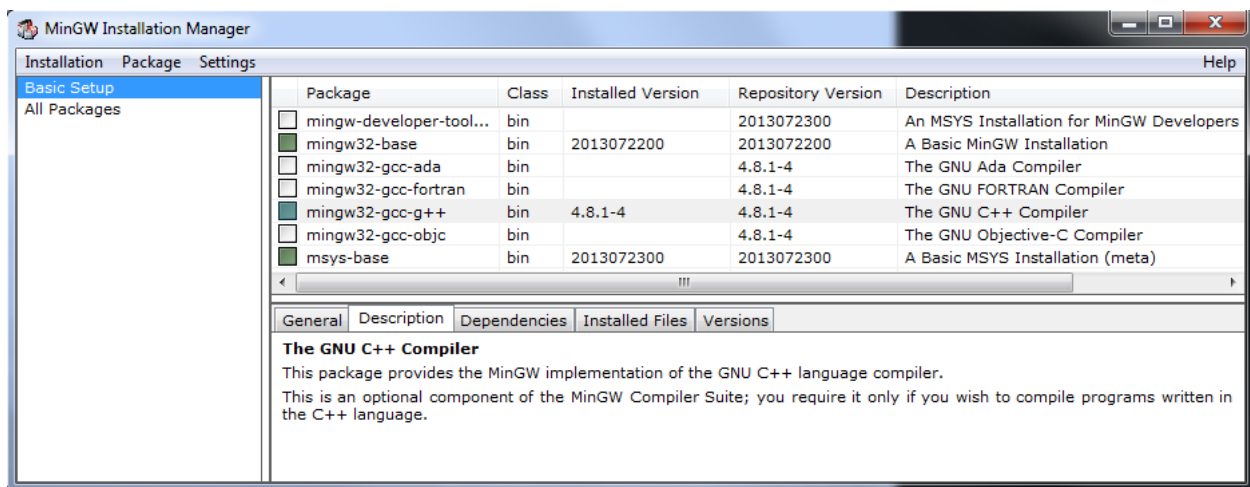- mingw32-gcc-g++

- msys-base

Figure 1: MinGW Installation Manager

Then run the menu item Installation -> Apply Changes. Click 'Apply' in the little Schedule of Pending Actions window that pops up. After all the packages download and install, you can close the MinGW Installation Manager window.

Next, bring up a DOS prompt. Do this by clicking the Windows button in the lower-left corner, typing `cmd.exe` into the search bar, and pressing enter. At the DOS prompt, execute the following commands:

```
cd C:\MinGW\bin
mingw-get update
mingw-get install msys-wget msys-unzip msys-man
```

This will install additional programs for MSYS that we will need later to install dependencies. When the install command is finished, close the DOS prompt window and move on to the next step.

2.3.3 Start MSYS

Open a new MSYS shell (not to be confused with a DOS shell!) by launching MSYS from the Start menu or desktop or by running `C:\MinGW\msys\1.0\msys.bat`. The MSYS shell looks like this:
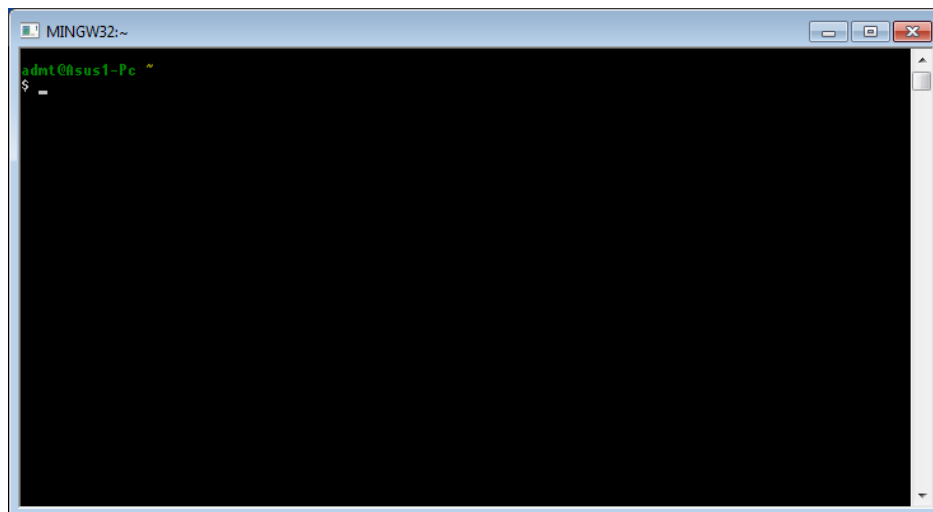
At the command prompt, run the command

```
$ /postinstall/pi.sh
```

Answer 'y' to continue the post install and then 'y' again that you have MinGW installed. Next, enter the path where you installed MinGW as you are instructed. When done, restart MSYS. At this point, installation of MinGW/MSYS is complete.

2.3.4 MSYS Tips

By default, the MSYS directory `C:\MinGW\msys\1.0` will be mounted as both the root directory `/` and as `/usr/`. The first time you run MSYS it will create a `home` directory in `C:\MinGW\msys\1.0`. The ~ (user home) directory will be something like `C:\MinGW\msys\1.0\home\<username>`. Traditional 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.

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/devel.zip.

Within the `.zip` file should be another directory named something like `AlloSystem-xxx`. You should move this directory to the directory `~/code/` you just made. After the directory `AlloSystem-xxx` 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 7.6.) Install dependencies with the following commands:

```
$ cd allocore
$ ./install_dependencies.sh
$ cd ..
```

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 commands:

```
$ make allocore
```

To check if the system was installed successfully you can build an example by first running

```
$ make gatherexamples
```

and then

```
$ make examples/allocore/av/simpleApp.cpp
```

If a sphere with an oscillating color shows up, then your system is configured 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. It is flexible enough to be used all the way from building simple prototypes to creating complex applications. This section describes how to setup and use AlloSystem's "build-and-run" capability.

By default, there are two directories recognized by the AlloSystem build-and-run facility: examples/ and work/. The directory examples/ contains example source code for libraries. The directory work/ is for user application source files and thus will be of primary concern here. You can place any source (.cpp) files having a main() function in the work/ directory (or any subdirectories thereof) and compile and run the source with the command

```
$ make work/myfile.cpp
```

Remember, you must be in ~/code/allosystem/ when running the make command. 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.,

```
$ make work/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/devel.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

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,

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

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

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

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

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 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. 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   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.

7.7   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 the line

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

to the beginning of C:\MinGW\msys\1.0\msys.bat where <myname> does not have spaces.

7.8 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.

7.9 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
```

7.10 '<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 Xcode is 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
```

7.11 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/`.