AlloSystem Install Tutorial

Lance Putnam and Lars Knudsen

February 2, 2013

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)
- Windows 7

For Windows users, it is highly recommended to install Linux Ubuntu. This can be done 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.

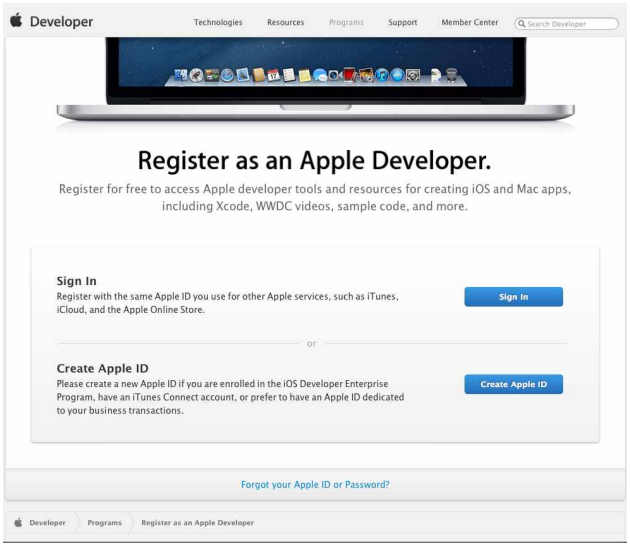
2.2 Mac OS X

Before Beginning

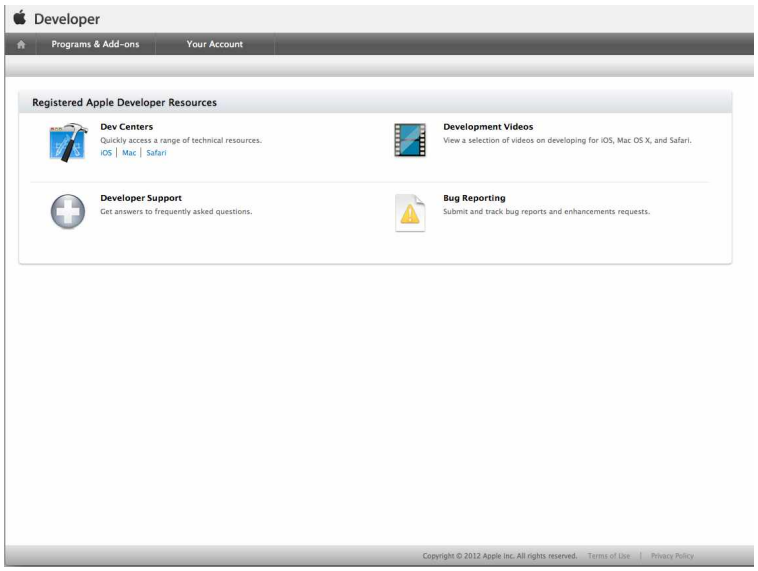
Before beginning to install the development tools, it is crucial that you complete any OS updates. If you don't, the installation of Xcode may fail with an "unknown error". (This is a known issue with OS X 10.6.x Snow Leopard and Xcode 3.2.6.) Do system updates by clicking the Apple symbol in the upper-left 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.

Get Apple Development Tools

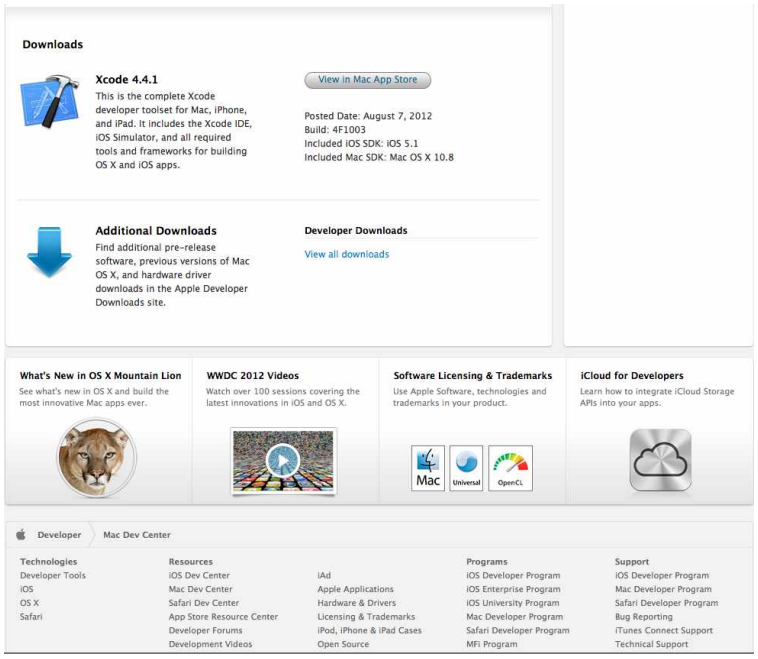
To get the Apple development tools, start by going to `http://developer.apple.com/programs/register`. Click the 'Sign In' button to login or create a new account.



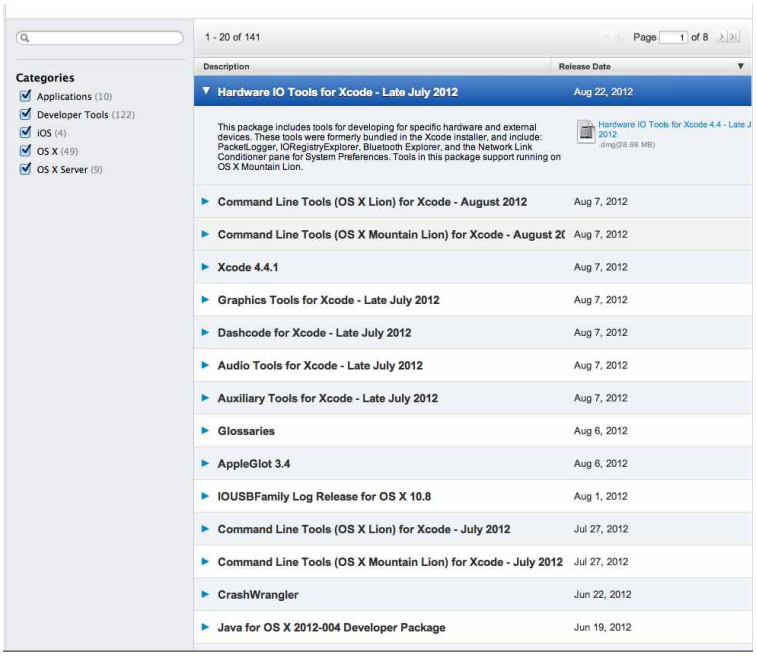
Under 'Dev Centers' click 'Mac'.



Scroll down and click 'View all downloads'.



This takes you to a search page.



Enter 'Xcode' into the search bar and get the latest version of *Xcode* **and** *Command Line Tools* that work for your version of OS X. (Note that with older versions of Xcode the command line tools are included with Xcode.) Unfortunately, every major version of OS X seems to have a new and backwards-incompatible version of Xcode. Use the following table to determine which version of Xcode and Command Line Tools to install with what version of OS X:

OS X Version	What To Install
10.5 (Leopard)	Xcode 3.1.4 Developer Tools
10.6 (Snow Leopard)	Xcode 3.2.6 and iOS SDK 4.3
10.7 (Lion)	Xcode 4.4.1, Command Line Tools (OS X Lion) - August 2012
10.8 (Mountain Lion)	Xcode 4.4.1, Command Line Tools (OS X Mountain Lion) - August 2012

Table 1: What tools to install for which version of OS X

After downloading the disk images (`.dmg` file), double-click them to mount them (if not done automatically). If you downloaded either Xcode 3.1.4 or Xcode 3.2.6, double-click the appropriate package (`.pkg`) file inside the mounted drive to run the installer. If you downloaded Xcode 4.4.1, then drag-n-drop the Xcode icon into your Applications folder and then run the Command Line Tools installer. When installation is complete, you will have both an editor and all the command-line tools we will be using such as `make` and g++.

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 (Leopard, Snow Leopard, Lion, or Mountain Lion).

2.3 Windows and MinGW/MSYS

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.

Install MinGW and MSYS

MinGW is an emulation of the GCC compiler suite. MSYS provides a Unix-like CLI and commands (bash, make, grep, etc.). Download and run the latest version of the MinGW installer mingw-get-inst-xxxxxxxx.exe from `http://sourceforge.net/projects/mingw/files/Installer/mingw-get-inst/`. Be careful when installing not to install MinGW in a path containing spaces, such as `C:\Program Files (x86)\MinGW`, as this will cause a lot of problems later on. The rest of the tutorial assumes `C:\MinGW` is the install directory. If you choose another install directory just replace any following instances of `C:\MinGW` with your path. When asked to "Select Components", choose

- C Compiler
- C++ Compiler
- MSYS Basic System

Finish the MinGW installer.

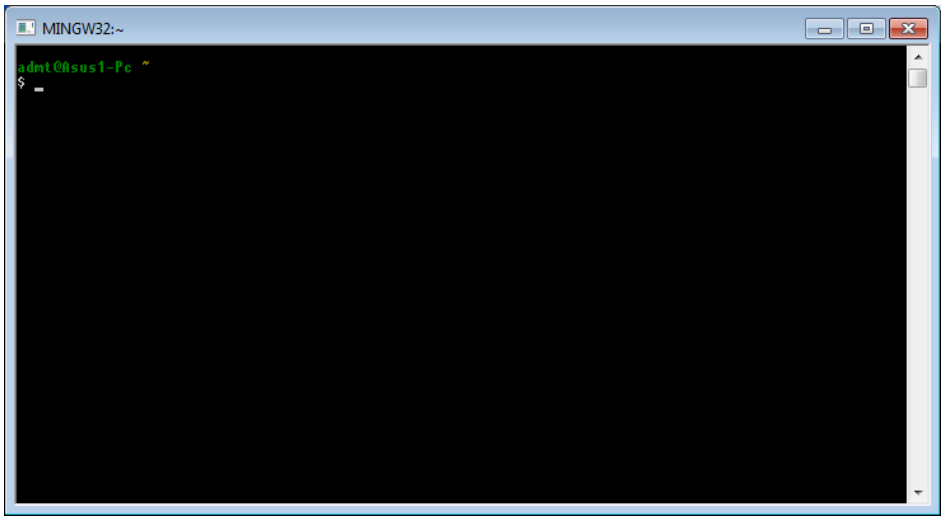
Next, bring up a DOS prompt. Do this by clicking the Windows button in the lower-left corner, typing `cmd.exe` into the search bar, and pressing enter. At the DOS prompt, execute the following commands:

```
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.

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:



By default, the MSYS directory `C:\MinGW\msys\1.0` will be mounted as root directory `/` and as `/usr/`. Traditional Windows drives, like `C:` and `D:`, can be accessed as `/d` or `/e`. 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. In the following instructions, when we say 'Unix shell' we mean MSYS for Windows users, and the 'Terminal' application for Mac OS X and Linux users.

3.1 Build AlloSystem

Download the AlloSystem source code `https://github.com/AlloSphere-Research-Group/AlloSystem/archive/stable13feb.zip`. Unzip the contents to `~/code/allosystem/` (if using Windows and MinGW/MSYS `~` will be something like `C:/MinGW/msys/1.0/home/<username>`). From the Unix shell, 'cd' into the `allosystem/` directory. If you do '`ls`' you should see something like:

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

Next, you need to install the necessary dependencies for AlloSystem. (Windows users only: you must deactivate any virus checkers. Many virus checkers are know to terminate important configuration processes that take place in the following steps. If you do not disable them, then the libraries will not get installed properly.) Install dependencies with the following commands (notice if you are using MinGW/MSYS then you don't need the sudo in front of the command):

```
$ cd allocore
$ sudo ./install_dependencies.sh
$ cd ..
$ cd alloutil
$ sudo ./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 please be patient. After dependencies are installed, the next step is to build the AlloCore and AlloUtil modules with the following commands:

```
$ make allocore
$ make alloutil
```

To check if the system is installed successfully you can build an example by navigating to the `~/code/allosystem/allocore` folder and run the following command:

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

If a ball with oscillating color shows up, then your system is configured correctly.

3.2 Build GLV and Gamma

GLV is a GUI toolkit and Gamma is an audio synthesis library. We will use these libraries to extend the functionality of AlloSystem. First, download the source code of each at `https://github.com/AlloSphere-Research-Group/GLV/zipball/master` and `https://github.com/LancePutnam/Gamma/zipball/master`. 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
```

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 Advanced Project Configuration

5.1 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 specify it generically as `[userdir]`. `[userdir]` 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]/`.

5.2 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`.

5.3 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/
```

6 Troubleshooting

6.1 '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.

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

If you get an error when trying to execute a Bash script (.sh file), then it is possible that its file mode is incorrect. Simply run

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

on the troublesome script. This will make it executable.

6.3 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/`.