HOW TO: Enable Monkey X with MinGW32 Cross Compiler

Table of Contents

If you are regularly building open source software and use both Microsoft Windows and Linux, then you will be in no doubt that it is much easier to build such software with the MinGW32 cross compiler in a Posix environment than it is to use MinGW32 with makefiles on Microsoft Windows.

On a side note, it's easier to build wxWidgets and wx applications for Microsoft Windows using a MinGW32 cross-compiler on Linux than it is to use the native Microsoft Windows MinGW32 compiler.

The purpose of this document is to guide you on how to set up a cross compile environment for Unix like systems to target Microsoft Windows operating systems, so that you can use such a set up for use with the MonkeyX compiler.

It will not cover due to restricted libraries and complexities :

- MS Windows to Linux. Though in theory; running the Linux version of transcc in Cygwin should be possible if all the dependencies are met.
- MS Windows to Apple OS X or Apple iOS
- Linux to Apple OS X or Apple iOS
- Apple OS X to Linux

For you to be able to compile MonkeyX applications using a cross compiler a number of thing need to be done first.

1. Install the native compiler tools chain.
2. Install QtCreator and the Qt Libraries so the TED can be rebuilt with a few modifications.
3. Install the MinGW cross compiler.
4. Install a Microsoft Windows version of OpenAL.
5. Install a compatible Microsoft Windows environment such as WINE for testing purposes.
6. Modify the Host systems version of the MonkeyX compiler transcc.
7. Modify the gcc_winnt Makefiles in the targets templates for glfw/glfw3.
8. Modify the hosts MonkeyX config file so that the MINGW toolchain can be used.
9. Modify the MonkeyX IDE TED to allow you to build both the native and Windows version.

You should have number one and maybe number two on the list above already installed.

I have included modified versions of MonkeyX 86c; transcc, TED, Makefiles and the config.txt

files.

You should download the latest version of MonkeyX from github or add the modifications to the compiler you wish to use.

HOW TO: Enable Monkey X with MinGW32 Cross Compiler

Linux

STEP ONE: Install a MinGW Cross Compiler

Currently there should be two versions of the MinGW32 cross compiler in the repositories. Theses are mingw32 and mingw-w64, with the latter be able to compile binaries for both 32bit and 64bit versions of Microsoft Windows. You should choose that latter if available as the compiler will be more up to date. Note that the mingw-w64 designated as i686 will only generate 32 bit binaries, while the one designated as x86_64 will generate 32 bit and 64 bit binaries. Unfortunately, version 4.8 has a bug that prevents the use of creating 32 bit binaries. So the work around is to install both the i686 and x86_64 version of mingw-w64 and switch tool chains in the configuration files.

Debian/Ubuntu/Linux Mint etc

```
sudo apt-get install mingw-w64
```

Fedora

```
sudo dnf install mingw64-gcc-c++ mingw32-gcc-c++
```

OpenSUSE

MinGW for OpenSUSE is not in the main official repositories. You will have to add a non official repository. The one I looked at was located at
http://download.opensuse.org/repositories/windows:/mingw:/win32/openSUSE_13.2/

```
zypper install mingw32-cross-gcc-c++
```

The above lines should install the core mingw-w64 C/C++ tool chain, development packages and binutils for both the i686 and x86_64 versions depending on the Linux distribution. It will not install a MinGW compatible debugger. If you need one then look for a package with a name similar to gdb-mingw-w64.

Depending on the distribution, the location for the MinGW tool chain and it's libraries can differ, but generally you should find them in

/usr/bin
/usr/i686-w64-mingw
/usr/x86_64-w64-mingw
/usr/local/i686-w64-mingw
/usr/i686-w64-mingw/sys-root/mingw
/usr/x86_64-w64-mingw/sys-root/mingw
/usr/lib/gcc/i686-w64-mingw
/usr/lib/gcc/x86_64-w64-mingw
/usr/lib64/gcc/i686-w64-mingw
/usr/lib64/gcc/x86_64-w64-mingw

STEP TWO: Install a Windows Version of OpenAL-Soft

You could use the Creative Labs version of OpenAL and copy it into one of the aforementioned Linux library paths, but the best thing is to either use the pre-compiled libraries or build an up to date OpenAL-Soft version. According to some sources scattered around the web. OpenAL-Soft is actually fast than what's on offer from Creative Labs, which to be frank has had any real updates from Creative in God knows how long.

To build OpenAL-Soft you will need the source code and the tool cmake. You can get the OpenAL-Soft source code from Sourceforge or Strangesoft, but the Surceforge version is a little out of date. http://kcat.strangesoft.net/openal.html, https://github.com/kcat/openal-soft
http://sourceforge.net/projects/openal-soft/

If you have downloaded the pre-compiled binaries, then they will need to be renamed OpenAL32.dll and moved into the bin directory of the MinGW tool chain or location within the system path. You could always modify your .profile in your home directory to include the locations of these files to the PATH variable.

You will find cmake in the distribution repositories where you can also download a GUI front end. To compile the OpenAL-Soft source for Microsoft Windows from our Linux host; we need to create what is know as a tool chain file to tell cmake what tools to use. Create an empty text file and copy and paste the text below and save it as **i686-w64-mingw32.cmake**. Note that you can create another of these files for the x86_64 tool chain, just alter the i686 parts to x86_64.

```
# the name of the target operating system
SET(CMAKE_SYSTEM_NAME Windows)

# which compilers to use for C and C++
SET(CMAKE_C_COMPILER i686-w64-mingw32-gcc)
SET(CMAKE_CXX_COMPILER i686-w64-mingw32-g++)
SET(CMAKE_RC_COMPILER i686-w64-mingw32-windres)

# here is the target environment located
SET(CMAKE_FIND_ROOT_PATH /usr/i686-w64-mingw32 /usr/local/i686-w64-mingw32)

# adjust the default behaviour of the FIND_XXX() commands:
# search headers and libraries in the target environment, search
# programs in the host environment
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

Extract the OpenAL-Soft source and change directory in the OpenAL-Soft root directory. You then need to execute cmake with the line below. If you intend to use both the i686 and x86_64 versions; then alter the `DCMAKE_INSTALL_PREFIX by changing the i686-w64-mingw32 for x86_64-w64-mingw32 for the version you wish to build etc.`

```
cmake -DCMAKE_TOOLCHAIN_FILE=/path-to-toolchain-file/i686-w64-mingw32.cmake
-DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local/i686-w64-mingw32
```

After cmake has done; you need to execute `make` and then `sudo make install`.

NOTES:

DCMAKE_TOOLCHAIN_FILE

The toolchain file path

DCMAKE_BUILD_TYPE

The build type configuration Debug,Release, RelWithDebInfo and MinSizeRel

DCMAKE_INSTALL_PREFIX

The location of where to install compiled code. The default is always /usr/local

If you see warnings about not finding DirectSound; then you will need to manually add the locations to -DDSOUND_INCLUDE_DIR and -DDSOUND_LIBRARY. You will find that there are some within the MinGW32 cross compiler.

STEP THREE: Install WINE

NOTE: WINE IS NOT A SANDBOX ENVIRONMENT. THIS MEANS THAT ANY MALWARE CAN AFFECT WHATS IN YOUR HOME DIRECTORY.

For you to test the programs you compile you will either have to transfer them over to a real Microsoft Windows operating system or install WINE for your distributions repository.

Once you have WINE installed and set up you will need to open a file called `system.reg`. You will find it within your home directory in the hidden .wine directory.

When you have got this file open, look for the section named `[System\\CurrentControlSet\\Control\\Session Manager\\Environment]` and append to the end of the variable string of "PATH" actual location of the compiler like the example below. Remember paths may be different on different Linux distributions and double back-slash are required for paths. Note drive Z: is mapped to the Linux root directory.

```
"PATH"=str(2):"C:\\windows\\system32;C:\\windows;C:\\windows\\system32\\wbem;C:\
\Program Files (x86)\\Dirac;Z:\\usr\\i686-w64-mingw32\\lib;Z:\\usr\\x86_64-w64-
mingw32\\lib;Z:\\usr\\lib\\gcc\\i686-w64-mingw32\\4.8;Z:usr\\lib\\gcc\\x86_64-
w64-mingw32\\4.8"
```

If you don't add the previous line, then expect to see a few DLL libray not found errors.

Apple Mac OS X

NOTE: Installing a MinGW compiler requires an Apple Mac with an Intel CPU.
Prerequisites: Any of the following to install WINE, WINE TRICKS and CMAKE.
Homebrew, MacPorts or Fink

STEP ONE: Install a MinGW Cross Compiler

You can download a working version of the MinGW-w64 tool chain for Apple Mac OS X from http://mingw-w64.org/doku.php/download/darwin or http://crossgcc.rts-software.org/doku.php

You can extract the MinGW archive to any location, but this guide will assume that you have extracted everything to your home directory and renamed the extracted directory as mingw-w32.

Once you have extracted the files you will need to edit/create a .bash_profile in your home directory to add the path to the new compiler to the system PATH variable. To do this you would be better of using the command line and the **nano** text editor.

Open a terminal (Applications → Utilities → Terminal) and type the command below

```
nano ~/.bash_profile
```

Enter the text below in to it. You can save the file by holding the Ctrl key and Pressing 'O'. You don't need to change the name, so just press the Enter/Return key and exit nano with Ctrl+X key combination.

```
MINGW=$HOME/mingw-w32
MINGW_GCC=4.9.0

MINGW_I686=$MINGW/i686-w64-mingw32:$MINGW/i686-w64-mingw/bin:$MINGW/i686-w64-
mingw/include:$MINGW/i686-w64-mingw/lib:$MINGW/libexec/gcc/i686-w64-
mingw32/$MINGW_GCC

MINGW_X86_64=$MINGW/x86_64-w64-mingw32:$MINGW/x86_64-w64-mingw/bin:
$MINGW/x86_64-w64-mingw/include:$MINGW/x86_64-w64-mingw/lib:
$MINGW/libexec/gcc/x86_64-w64-mingw32/$MINGW_GCC

export PATH=$MINGW:$MINGW/bin:$MINGW/include:$MINGW/lib:$MINGW/libexec:
$MINGW_I686:$MINGW_X86_64:$PATH:/usr/local/sbin
```

SPECIAL NOTE:
You should change the value of the MINGW_GCC variable if you are using a different version of the MinGW gcc tool chain.

After you have made these changes you should log out and back in or restart your system.

STEP TWO: Install a Windows Version of OpenAL-Soft

You could use the Creative Labs version of OpenAL and copy it into one of the aforementioned Linux library paths, but the best thing is to either use the pre-compiled libraries or

build an up to date OpenAL-Soft version. According to some sources scattered around the web. OpenAL-Soft is actually fast than what's on offer from Creative Labs, which to be frank has had any real updates from Creative in God knows how long.

To build OpenAL-Soft you will need the source code and the tool cmake. You can get the OpenAL-Soft source code from Sourceforge or Strangesoft, but the Surceforge version is a little out of date. http://kcat.strangesoft.net/openal.html, https://github.com/kcat/openal-soft
http://sourceforge.net/projects/openal-soft/

If you have downloaded the pre-compiled binaries, then they will need to be renamed OpenAL32.dll and moved into the bin directory of the MinGW tool chain or location within the system path. You could always modify your .bash_profile/.profile in your home directory to include the locations of these files to the PATH variable.

For cmake, do not use the one from the cmake website as it will not work correctly. You should instead install the one provided by either Homebre, MacPorts or Fink. Please read the documentation on the respective tools website documentation for installation and set up.

Before we can build the software; we need to create what is know as a tool chain file to tell cmake what tools to use. Create an empty text file and copy and paste the text below and save it as **i686-w64-mingw32.cmake**. Note that you can create another of these files for the x86_64 tool chain, just alter the i686 parts to x86_64.

```
	# the name of the target operating system
	SET(CMAKE_SYSTEM_NAME Windows)

	# which compilers to use for C and C++
	SET(CMAKE_C_COMPILER i686-w64-mingw32-gcc)
	SET(CMAKE_CXX_COMPILER i686-w64-mingw32-g++)
	SET(CMAKE_RC_COMPILER i686-w64-mingw32-windres)

	# here is the target environment located
	SET(CMAKE_FIND_ROOT_PATH $ENV{HOME}/mingw-w32 $ENV{HOME}/mingw-w32/i686-w64-
mingw32)

	# adjust the default behaviour of the FIND_XXX() commands:
	# search headers and libraries in the target environment, search
	# programs in the host environment
	set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
	set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
	set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

Extract the OpenAL-Soft source and change directory in to the OpenAL-Soft root directory. You then need to execute cmake with the line below. If you intend to use both the i686 and x86_64 versions; then alter the `DCMAKE_INSTALL_PREFIX by appending i686-w64-mingw32 or x86_64-w64-mingw32 for the version you wish to build.`

```
cmake -DCMAKE_TOOLCHAIN_FILE=/path-to-toolchain-file/i686-w64-mingw32.cmake
-DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=$HOME/mingw-w32
```

After cmake has done; you need to execute `make` and then run `make install`.

NOTES:

DCMAKE_TOOLCHAIN_FILE

The toolchain file path

DCMAKE_BUILD_TYPE

The build type configuration Debug,Release, RelWithDebInfo and MinSizeRel

DCMAKE_INSTALL_PREFIX

If you see warnings about not finding DirectSound; then you will need to manually add the locations to -DDSOUND_INCLUDE_DIR and -DDSOUND_LIBRARY. You will find that there are some within the MinGW32 cross compiler.

STEP THREE: Install WINE

NOTE: WINE IS NOT A SANDBOX ENVIRONMENT. THIS MEANS THAT ANY MALWARE CAN AFFECT WHATS IN YOUR HOME DIRECTORY.

For you to test the programs you compile you will either have to transfer them over to a real Microsoft Windows operating system or install WINE from either Homebrew, MacPorst or Fink. Please read the documentation on the respective tools website documentation for installation and set up.

Once you have WINE installed and set up you will need to open a file called `system.reg`. You will find it within your home directory in the hidden .wine directory.

When you have got this file open, look for the section named `[System\\CurrentControlSet\\Control\\Session Manager\\Environment]` and append to the end of the variable string of "PATH" actual location of the compiler and libraries like the example below. Remember paths may be different on different Linux distributions and double back-slash are required for paths. Note drive Z: is mapped to the Mac OS X root directory.

```
"PATH"=str(2):"C:\\windows\\system32;C:\\windows;C:\\windows\\system32\\wbem;Z:\
\Users\\jason\\mingw-w32\\libZ:\\Users\\jason\\mingw-w32\\i686-w64-mingw32\\lib
Z:\\Users\\jason\\mingw-w32\\x86_64-w64-mingw32\\lib"
```

If you don't add the previous line, then expect to see a few DLL libray not found errors.