BuildingWindows

Warning: The build instructions listed here only apply to Mumble v1.3.x or older. If you want to build Mumble starting from v1.4.0, checkout https://github.com/mumble-voip/mumble/tree/master/docs/dev/build-instructions

Building Guides

This page is one of a set of Building pages/guides for the different OSes with information on building Mumble/Murmur.

BuildingLinux | BuildingFreeBSD | BuildingOpenBSD | BuildingMacOSX | BuildingWindows

Contents

Commandline instructions

Whenever something appears

```
like this
```

you're supposed to enter it in that command shell (or copy it from this webpage and right click in the command window and select *Paste*).

Note that *each line* is a separate command. So, if you wanted to do the following,

```
cd mumble
nmake
```

you would type "cd mumble" in your command prompt, and press enter, and then you would type "nmake", and press enter.

Ok. So you're ready to start working.

Build using MSVC on Windows

Introduction

Mumble has quite a few dependencies for building on Windows, and as the feature set grows, so does the list of dependencies. Therefore we have built some automated scripts to create a sane build environment for Mumble. To make this build environment as similar as possible across all of Mumble's supported Platforms, the Windows build is strongly dependent on Cygwin (https://www.cygwin.com/).

The mumble-releng (https://github.com/mumble-voip/mumble-releng) Github repository has an up-to-date README (https://github.com/mumble-voip/mumble-releng/blob/master/buildenv/1.3.x/win32-static/README) on how to create the build environment!

Note:
The most up-to-date information on how to create a Mumble build environment are always to be found in this README file!

Elaboration: There is an older deprecated version of this article at BuildingWindows (deprecated) showing how to build Mumble manually with QT4 which is far more complex.

Software you will need

Note:
We currently depend on Visual Studio to be in their default locations, and the Windows 7 SDK to be in *C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A* (rather than the default *C:\Program Files\...*).

- *Visual Studio Community 2015 Update 3*: You need MSVC 2015. During installation you can deselect the Windows SDKs, because we are going to install the up-to-date one, listed below.
- Windows SDK 7 (https://www.microsoft.com/en-us/download/details.aspx?id=8279) for XP/x86 Overlay (You may have to temporarily remove *Microsoft Visual C++ 2010 * Redistributable* for the installer to work (http://stackoverflow.com/questions/19366006/error-when-installing-windows-sdk-7-1#23032807).)
- Windows SDK (https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk) (If you are on Windows 7, don't select "Windows Performance Toolkit", because it isn't compatible)
- Git (https://msysgit.github.io/): You can use any version of Git, it only has to be in your PATH variable.

Creating the build environment

First you need to clone the mumble-releng repository into a directory in which your Windows user has write access. *cd* into that directory, then

```
git clone https://github.com/mumble-voip/mumble-releng.git mumble-releng
```

In the terminal window, change to the directory the scripts for Windows (32-bit) and static build reside in.

```
cd mumble-releng/buildenv/1.3.x/win32-static
```

and execute

```
setup.cmd
```

This should install a Mumble build environment in your *C:\MumbleBuild* directory named with a date and shortened commit hash like *win32-static-1.3.x-2015-05-25-1234ab7*.

If everything went right Windows Explorer should open up the new build environment directory as named above.

Build Mumble's dependencies

To start building Mumble's dependencies, double-click the "MumbleBuild - cygwin" shortcut. This will open a cygwin terminal. To change the current directory to the *build environment's local clone* of the mumble-releng repository type:

```
cd mumble-releng/buildenv/1.3.x/win32-static
```

Note:
This mumble-releng directory is a copy. Should you want to update your build environment in the future, proceed from the original.

To download and build all dependencies (this will take a long time!), execute

```
./build-all.bash
```

Once all dependencies are built, you will be returned to your Cygwin shell. Make sure it did not stop because of an error.

The built dependencies are located in the corresponding *.build* folder.

Note:
If you are using Visual Studio on WindowsOS, Please notice that sometimes the install process would not add itself into PATH. It may cause "protobuf.build" cant finish as expected.

Note:
If you are using a non-English version of Visual Studio on WindowsOS, Please check the page above. The best way to solve is install a Visual Studio English language pack AND set your sysyem language into English. QTBUG-56388 (https://bugreports.qt.io/browse/QTBUG-56388)

Building Mumble

To build Mumble itself using your newly-built build environment, you should use the "MumbleBuild - cmd" shortcut to launch a Windows command prompt.

Next, make a checkout of the Mumble source tree and enter the root of the source tree with

```
git clone --recursive https://github.com/mumble-voip/mumble.git mumble
cd mumble
```

to generate Makefiles for use with the build environment, run

```
qmake -recursive main.pro CONFIG+="release static no-elevation no-g15 no-asio" CONFIG-=sse2
```

Note:
This build environment does not install the libraries and headers needed to build Mumble with G15 LCD or ASIO audio support. If you want to build Mumble with one of them you need to install them to the build environment on your own. In this case remove *no-g15* or rather *no-asio* accordingly.

After all this preparation start the actual Mumble compilation with

```
nmake
```

You should end up with a mumble.exe and a murmur.exe binary (and a whole collection of .DLL files to go along with the two) in the "release" directory in the root of the Mumble source tree.

If you want to distribute your own mumble.exe you can either create an installer (see below) or collect mumble.exe and the needed .DLL files yourself.

(Optional) Visual Leak Detector

Note:
VLD is no longer a default requirement for a Mumble build environment. You only have to install it if you want to use it in which case you have to manually enable it with *CONFIG+=vld*.

Download (http://dmoulding.googlepages.com/vld) VLD and install it to its default install location.

If you're using Visual C++ Express Edition, you will need to manually extract the files using a tool like 7-zip (http://7-zip.org/). Extract it to *C:\dev* and adjust the VLD_PATH to point to it.

Note:
VLD is only enabled for debug builds. If you only compile Release builds you do not need it.

(Optional) Custom Dependency Paths

The build files were modified to support custom dependency paths a while ago. This is for the people who have **the dependencies installed in some other place than the autogenerated directory structure**. To specify the custom paths you need to **create a *winpaths_custom.pri* file** to the root of your Mumble project. In this file you can override all paths found in *winpaths_default.pri*. For example:

```
OPENSSL_PATH = C:\\dev\MyOpenSSLIsSomewhereElse
ICE_PATH = C:\\Program Files (x86)\\ZeroC\\Ice-3.4.1
```

Would make the build process search its OpenSSL and Ice dependencies in the specified folders and use defaults for everything else. Note that you should only override the variables for dependencies you actually installed in non-default locations to prevent clashes with possible future updates.

Note: If you copied *winpaths_default.pri* to create your *winpaths_custom.pri* make sure to delete the following lines from your *winpaths_custom.pri* file:

```
# Include custom file if it exists
exists(winpaths_custom.pri) {
```

```
        include(winpaths_custom.pri)
}
```

(Optional) Build a Mumble installer package

If you want to create an installable .msi package from your self-compiled Mumble some additional steps are needed.

Download and install the latest WIX stable Version (currently 3.8) from here (http://wixtoolset.org/releases/).

Set the following environment variables as needed (see defaults in installer/Settings.wxi):

```
MumbleSourceDir  default: \dev\mumble
MumbleQtDir  default: \dev\QtMumble
MumbleDebugToolsDir  default: C:\Program Files (x86)\Debugging Tools for Windows (x86)
MumbleSndFileDir  default: \Program Files (x86)\Mega-Nerd\libsndfile\bin
  Define MumbleNoSndFile to exclude  libsndfile
MumbleMySQLDir  default: \dev\MySQL
  Define MumbleNoMySQL to exclude  MySQL
MumbleIceDir  default: \Program Files (x86)\ZeroC\Ice-3.4.2\bin\vc100
  Define MumbleNoIce to exclude Ice
MumbleOpenSslDir  default: \dev\openssl
MumbleZlibDir  default: \dev\zlib
MumbleMergeModuleDir  default: C:\Program Files (x86)\Common Files\Merge Modules
Define MumbleSSE to include SSE
Define MumbleNoSSE2 to exclude SSE2
Define MumbleNoG15 to exclude G15
```

Open installer/MumbleInstall.sln, switch to release and build the installer.

Once this completed successfully run the *build_installer.pl* script to include all translations into your installer.

You should now have a working .msi installer.

Build using MXE on Debian and derivates

Note:
Mumble supports MinGW since commit 10079ed9867308aad098231f86e260bd831b0ac6 (https://github.com/mumble-voip/mumble/commit/10079ed9867308aad098231f86e260bd831b0ac6) (March 2017, first stable version 1.3.0).

Introduction

It is possible to cross-compile Mumble using a MinGW toolchain. If you're more comfortable with a Unix-like system, you will probably prefer this method. The easiest way to cross-compile Mumble is by using MXE, which provides all the required dependencies.

Features currently not available with MinGW:

- Overlay
- Logitech G15 LCD
- ZeroC Ice
- Bonjour

Note:
In the guide there are some commands with *${ARCH}* in them.

You need to replace it with the desired architecture, which can be *x86_64* (64 bit) or *i686* (32 bit).

(If you are on a 64bit system and the above doesn't work for you, you might want to try *x86-64* (with a dash instead of an underscore) instead).

Installing the required MXE packages

Add MXE's repository to your system's sources:

Debian

```
echo "deb https://dl.mumble.info/mirror/pkg.mxe.cc/repos/apt stretch main" | sudo tee /etc/apt/sources.list.d/mxe.list
```

Ubuntu

```
echo "deb https://dl.mumble.info/mirror/pkg.mxe.cc/repos/apt xenial main" | sudo tee /etc/apt/sources.list.d/mxe.list
```

Add the repository's key:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 86B72ED9
```

Update packages index:

```
sudo apt update
```

Install the required packages

```
sudo apt install \
        mxe-${ARCH}-w64-mingw32.static-qtbase \
        mxe-${ARCH}-w64-mingw32.static-qtsvg \
        mxe-${ARCH}-w64-mingw32.static-qttools \
        mxe-${ARCH}-w64-mingw32.static-qttranslations \
        mxe-${ARCH}-w64-mingw32.static-boost \
        mxe-${ARCH}-w64-mingw32.static-protobuf \
        mxe-${ARCH}-w64-mingw32.static-sqlite \
        mxe-${ARCH}-w64-mingw32.static-flac \
        mxe-${ARCH}-w64-mingw32.static-ogg \
        mxe-${ARCH}-w64-mingw32.static-vorbis \
        mxe-${ARCH}-w64-mingw32.static-libsndfile
```

Preparing the environment

Clone Mumble's Git repository

```
git clone https://github.com/mumble-voip/mumble.git
cd mumble
```

Get the ASIO SDK

```
wget https://www.steinberg.net/sdk_downloads/asiosdk2.3.zip -P /tmp/
unzip /tmp/asiosdk2.3.zip -d /tmp/
mv /tmp/ASIOSDK2.3 3rdparty/asio
```

Export environment variable to tell QMake where MXE's protobuf compiler is

```
export MUMBLE_PROTOC=/usr/lib/mxe/usr/x86_64-pc-linux-gnu/bin/protoc
```

Add MXE's directory to PATH

```
PATH=$PATH:/usr/lib/mxe/usr/bin
```

Build

Run QMake to process the project(s) files

```
${ARCH}-w64-mingw32.static-qmake-qt5 -recursive CONFIG+="release g15-emulator no-overlay no-bonjour no-elevation no-ice"
```

Start the build

```
make
```