

LA COMPILACION DE MIDIEDITOR BAJO WINDOWS

by F.MUÑOZ “ESTWALD”

Acerca de MidiEditor Custom

MidiEditor Custom no es solamente una versión no oficial del MidiEditor de Marcus Schwenk.

Hace tiempo que se incorporaron mejoras como soporte de la librería Fluidsynth, soporte para plugins VST, caja de ritmos, secuenciador y muchos otros detalles que van un poco mas allá, pero a pesar de los cambios y añadidos, se conserva la esencia del programa original.

La razón de este documento es explicar el proceso para poder llevar a cabo la compilación del programa bajo el entorno de programación Qt y sistema operativo Windows - que en origen era compilado con la versión 5.15.2 - para portarlo a la 6.9.0 actual, añadiendo en el proceso la librería Fluidsynth, en lugar de utilizar la librería oficial ya construida.

ABI, Shared y Static

La Application Binary Interface (ABI) define como los módulos de código interactúan a nivel binario. Un ABI diferente hace que una librería no sea reconocida, pero también el compilar código con diferentes compiladores, puede provocar la inestabilidad de los programas.

Las librerías pueden ser compartidas o estáticas. Una librería compartida puede ahorrar espacio de almacenamiento si se comparte por varios programas y también permite poder actualizarla o corregir errores, sin la necesidad de compilar de nuevo los programas, pero:

- 1) Si las librerías compartidas son muchas y solo se utilizan para un programa, el ahorro de espacio no es tal y encima existe el problema de tener que añadirlas para su distribución con problemas de copyright, etc
- 2) Si se combinan desde diferentes compilaciones, pueden compartir librerías comunes pero con diferentes símbolos para las funciones y... error!
- 3) Una compilación estática permite un ejecutable con todo contenido.

Hasta ahora Midieditor Custom usaba librerías compiladas en estático que (las mencionadas Qt 5.15.2), que salvaban además, el problema de colisionar con las DLL de Fluidsynth. Sin embargo, al menos en la versión de 32 bits, se apreciaban problema de estabilidad debido a la diferencia de compiladores usados.

Qt 6.9.0

Para bajarse el entorno de compilación Qt, requiere de registrarse en su página oficial:

<https://login.qt.io/login>

Hazlo como particular para desarrollo open source.

<https://www.qt.io/download-open-source>

<https://www.qt.io/download-dev>

Desde ahí puedes bajar el instalador y una vez iniciado podras elegir Qt 6.9.0 y las tools necesarias para compilar con librerías compartidas. Solo se suministra versión para 64 bits. Por lo que si se precisa 32 bits por alguna cuestión de compatibilidad... tendrás que compilar Qt 6.9.0 tu mismo para esa versión.

En tools los compiladores suministrados son MinGW 13.1.0 de 64bits con lo justo y necesario para compilar en el entorno Qt... pero no le pidas más, como por ejemplo, poder compilar los fuentes de FluidSynth (eso requiere librerías y herramientas que no se nos proporcionan). Si no quieres utilizar el entorno de desarrollo, tambien es posible bajarse el código fuente de Qt 6.9.0 sin registrarse y yo mismo te proporciono una herramienta, como verás mas abajo, que hace todo el proceso de bajar, configurar y compilar.

MSYS2

Yendo al grano, lo que se necesita es un entorno bajo Windows, que soporte las herramientas necesarias para compilar como librerías estática Qt 6.9.0, para 64 bits y 32 bits, que nos proporcione los compiladores y otras herramientas necesarias tanto para esto, como para Fluidsynth. Y que, además, nos proporcione librerías ya compiladas para ese entorno en estático, que nos evite tener que compilar muchos fuentes (solo necesitaremos compilar libinstpatch). MSYS2 cumple con las condiciones necesarias:

<https://www.msys2.org/>

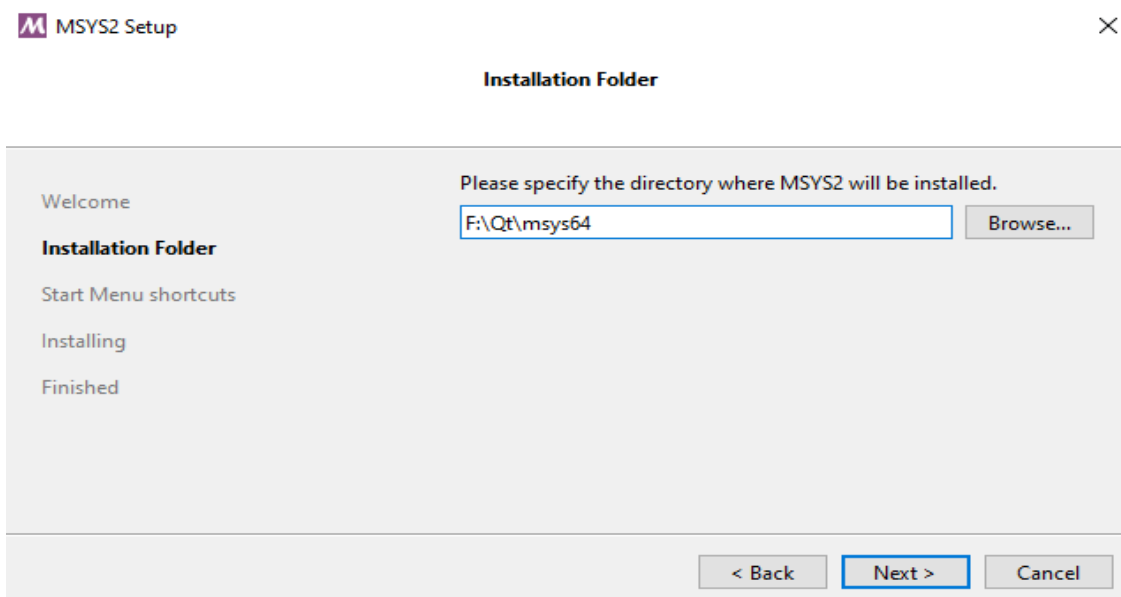
Pasos:

Antes de nada, tengo que comentar que requiere una cifra de espacio en el disco duro cercana a 60 GB libres. Solo el código compilado de Qt 6.9.0 (sin el módulo de Qtwebengine), ocupa la friolera de 34 GB... y hablamos de miles de ficheros pequeños, por lo que mi recomendación es usar una unidad SSD. Una vez compiladas las librerías puede liberarse el espacio que ocupan los objetos construidos y también el código fuente descomprimido (el comprimido en tar.gz, ocupa algo más de 1GB, no demasiado espacio dentro de lo que cabe)

- Descarga el instalador desde aquí:

https://github.com/msys2/msys2-installer/releases/download/2025-02-21/msys2-x86_64-20250221.exe

- Lanzalo e instala en el directorio Qt que corresponda. Por ejemplo, en mi caso F:\Qt\msys64 :



- Lo hacemos así porque esta instalación de MSYS2 es solo y exclusivamente para utilizarla con Qt. Si necesitas MSYS2 para otra cosa, haz otra instalación, porque esta la vamos a personalizar para que no de errores al compilar lo que necesitamos para Midieditor Custom.

- Una vez instalado, es preciso establecer la variable de entorno **QTENV** de esta forma:
QTENV=F:/Qt (porque en mi caso está en la unidad F:. Y usa barra de dividir /, recuerda).

Busca como añadir variables de entorno en tu sistema windows si no sabes como hacerlo.

CONFIGURAR MINGW32/64

Las instrucciones sirven igual para ambas versiones, pues ya se encargan las herramientas que proporciono de detectar el bash desde donde se lanzan.

- Entra en el directorio de instalación de MSYS2 (en mi caso F:\Qt\msys64) con el explorador de archivos.

Ahí veras los lanzadores de bash mingw32.exe y mingw64.exe. No entres en otro bash!. Lancemos pues, mingw64.exe para crear el entorno necesario para 64bits.

Bien, veremos una pantalla asi:



Escribimos:

\$ pacman -Syu

Nos actualizará algunas cosas, si nos pregunta respondemos 'Y' y seguramente, cuando termine cierre el bash. Si ocurre, volvemos a lanzarlo y ejecutamos:

\$ pacman -Su

- Entramos desde el explorador de archivos en el directorio F:\Qt\msys64\home\<usuario> (en mi caso F:\Qt\msys64\home\paco_)

- Copiamos dentro los ficheros **msys2tool.sh** y **qtmsys.sh** que encontraremos en los fuentes de Midieditor Custom en el directorio midieditor\building\MSYS2 Tools

-Ahora si hacemos en el bash de MinGW64:

\$ dir

deberiamos verlos:

```
paco_@DESKTOP-NL4R0CR MINGW64 ~  
$ dir  
msystool.sh qtmsys.sh  
  
paco_@DESKTOP-NL4R0CR MINGW64 ~$
```

msystool.sh

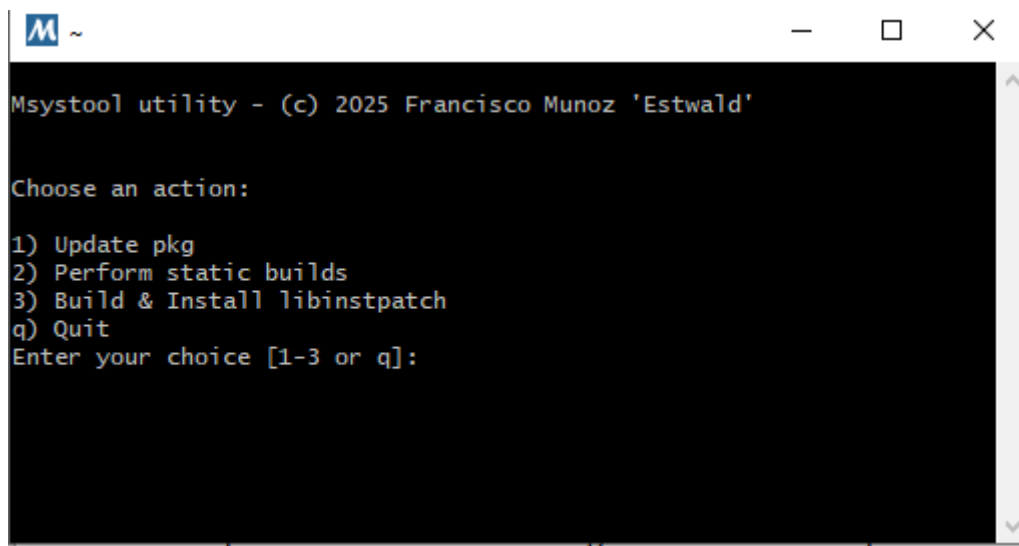
Esta herramienta consta de varias fases en las cuales actualizará MinGW64 (porque es el bash que hemos lanzado) con los compiladores, herramientas y librerías necesarias para poder compilar tanto Qt 6.9.0 como Fluidsynth con todos los componentes que nos hacen falta. Tambien aplicará parches y moverá alguna cosa para hacer que el uso de cmake en las distintas configuraciones, no acabe compilado en modo shared en lugar de static: se toma por defecto shared ante cualquier nimiedad.

NOTA: Antes de hacer nada, recuerda **que es preciso establecer la variable de entorno QTENV** de esta forma: QTENV=[F:/Qt](#)

- Lanzamos la herramienta de la siguiente manera:

\$./msystool.sh

y nos mostrará una pantalla tal que así:

A screenshot of a terminal window titled "M ~" with standard window controls. The terminal displays the text "Msystool utility - (c) 2025 Francisco Munoz 'Estwald'" followed by "Choose an action:". Below this, a list of options is shown: "1) Update pkg", "2) Perform static builds", "3) Build & Install libinstpatch", and "q) Quit". The prompt "Enter your choice [1-3 or q]:" is at the bottom of the list. The terminal has a dark background with light-colored text.

Hay que elegir todas las opciones de la 1 a la 3. No se automatizan todas las fases por si surgen errores imprevistos o se quiere realizar una de las subpartes. No se recomienda ejecutar mas de una vez cada opcion.

1) update pkg

Se instalan (se baja de internet, así que es obvio que requiere de conexión a internet) todos los paquetes necesarios, desde compiladores, pasando por utilidades y terminando por librerías necesarias. **Una vez realizado este paso, no se debería volver a realizar: recuerda que cualquier actualización podría hacer que la utilidad no funcionara de forma apropiada.**

2) Perform static builds

Aquí lo primero que se hace, es clonar el directorio **/mingw64/lib** en el directorio **/mingw64/lib-orig** para tener una copia sin alterar de dicho directorio (si hiciera falta, con renombrar las carpetas tendríamos de las librerías sin alterar).

Luego se procede a buscar todas las librerías shared, es decir, con **.dll.a** y moverlas a la carpeta **/mingw64/lib/sharedlibs**. Eso nos permite poder enlazarlas de forma manual, si las necesitamos.

Después se crea el directorio **/mingw64/lib/pkgconfig-static** donde se procederá a copiar y parchear los ficheros **.pc** de librerías como glib-2.0 o sndfile, requeridos por Fluidsynth. Este directorio **/mingw64/lib/pkgconfig-static** se añadirá de forma que se encuentren estos ficheros parcheados antes que los que están en **/mingw64/lib/pkgconfig**, engañando a cmake.

3) Build & Install libinstpatch

Como su nombre indica, primero se baja el código fuente de libinstpatch, lo configura, lo compila y lo instala procediendo a parchear lo que sea necesario.

q) Quit

Salida de la aplicación.

Ahora ya tenemos todo preparado para poder bajar los fuentes y compilar las librerías **Qt 6.9.0** mediante la segunda aplicación: **qtmsys.sh**. Recuerda que si necesitas las librerías de 32 bits, tienes que lanzar el bash **mingw32.exe** y hacer los mismos pasos (la única diferencia es que se aplica sobre **/mingw32**, no sobre **/mingw64**).

qtmsys.sh

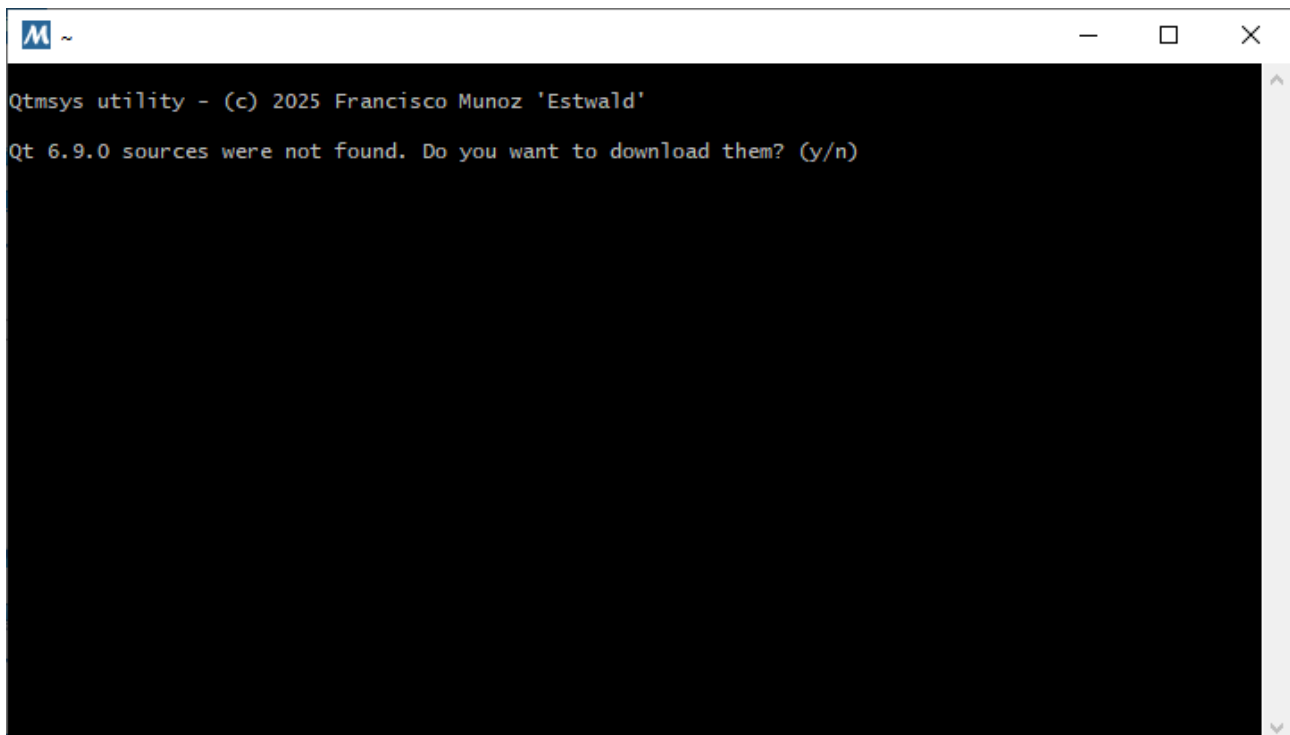
Esta es la herramienta que bajará el código fuente de **Qt 6.9.0**, parcheará algunas cosas, lo configurará, compilará e instalará para poder utilizarlo para compilar **Midieditor Custom**.

Antes de nada, recordar que la variable de entorno **QTENV** debe estar fijada. Y por supuesto, primero tienes que haber completado todos los pasos de **msystool.sh**.

Para lanzar la aplicación hacemos en el bash correspondiente:

```
$ ./qtmsys.sh
```

Al hacerlo, comprobará si el directorio con los fuentes existe (se instalan en **\$QTENV/qt-everywhere-src- 6.9.0**, que en mi caso sería **F:/Qt/qt-everywhere-src- 6.9.0**). Si no existe lo primero que veremos es esta pantalla para descargar:



```
Qtmsys utility - (c) 2025 Francisco Munoz 'Estwald'
Qt 6.9.0 sources were not found. Do you want to download them? (y/n)
```

Seleccionamos 'y' para descargar. Cuando termine de hacerlo, se iniciará la descompresión de los fuentes en un directorio temporal para prevenir fallos, luego borrará el código fuente de Qtwebengine (ocupa mucho espacio y requiere compiladores de Microsoft) y a continuación se parchearán algunos archivos:

qt-everywhere-src-6.9.0/qtquick3d/src/3rdparty/assimp/src/code/Common/DefaultIOStream.cpp:

se añade **#define _CRTIMP** antes de los **#include** para que la compilación sea estática.

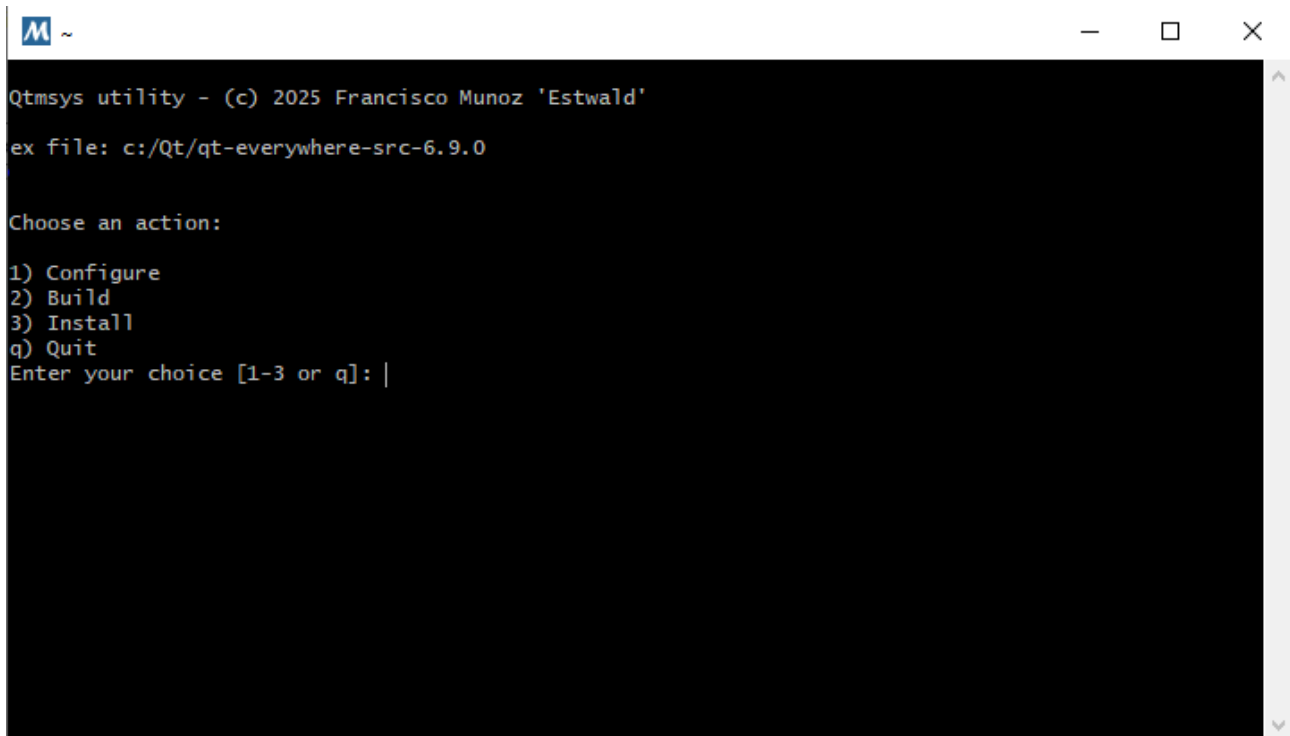
qt-everywhere-src-6.9.0/qtquick3d/tools/balsam/CmakeLists.txt

qt-everywhere-src-6.9.0/qtquick3d/tools/balsamui/CmakeLists.txt

qt-everywhere-src-6.9.0/qtquick3d/tools/meshdebug/CmakeLists.txt:

se añade la línea: **target_link_libraries(\${target_name} PRIVATE ucrt)** pa que encuentre la librería.

Una vez realizados estos pasos (o detectados que se realizaron correctamente) aparece este menu: (ver en página siguiente)



```
Qtmsys utility - (c) 2025 Francisco Munoz 'Estwald'
ex file: c:/Qt/qt-everywhere-src-6.9.0

Choose an action:
1) Configure
2) Build
3) Install
q) Quit
Enter your choice [1-3 or q]: |
```

1) Configure

Configura **Qt 6.9.0** de manera apropiada para ser compilada las librerías que lo componen. El proceso se mueve provisionalmente el directorio **/mingw64/lib/cmake** para que cmake al procesar el **CMakeList.txt** no detecte cosas que no nos convienen. Cmake mira en los directorios mencionados de cmake y pkgconfig, así como detecta las librerías compartidas (**dll.a**) y a la mínima, salta nuestra compilación estática por los aires.

Para mantener el código fuente libre de polvo y paja, se crea el directorio **\$QTENV/build-qt6-static**. Mucho cuidado con esto porque es el mismo directorio tanto para la compilación de 32 bits como la de 64 bits... la configuración borra el contenido anterior antes de iniciarse, por lo que no debería haber problemas si se hacen los pasos correctamente.

2) Build

Como sugiere inicia la compilación y construye todas las librerías y herramientas necesarias de Qt 6.9.0. Los compiladores que se instalan en MSYS2 son Mingw 15.1.0, actualmente, por lo que está mas actualizado que el de las tools de Qt.

3) Install

Una vez compilado todo sin errores (tarda horas...) solo queda instalarlo en el directorio destino-

Para la versión 32 bits sería:

\$QTENV/6.9.0/mingw_msys32_static

Y para la versión 64 bits sería:

\$QTENV/6.9.0/mingw_msys64_static

Y con esto ya tendríamos nuestras librerías estaticas necesarias para poder compilar con **Qt 6.9.0**

Compilar Fluidsynth

A pesar de que mi objetivo inicial era la compilación de las librerías en estático, por simplificación y posibilidad de actualizar por parte del usuario, la herramienta que he creado para compilar Fluidsynth permite crearla como librería estática o como compartida, pero eso si, con una única DLL. Por defecto, midieditor.pro está configurado para usar esta librería compartida.

Existen dos versiones de la misma, una de 32 bits y otra de 64 bits que lanzan desde fichero .bat los bash con los .sh correspondientes.

Se encuentra en midieditor\building con los nombres de **build_fluidsynth32bits.bat** y **build_fluidsynth64bits.bat** y el subdirectorio **build-fluidsynth** contiene los correspondientes **build-fluidsynth32bits.sh** y **build-fluidsynth64bits.sh** que se encargarán de bajar el código fuente, compilarlo e instalarlo para cada una de las versiones.

Igual que con las otras herramientas, **es necesario fijar la variable de entorno QTENV**. Por es así como encuentra los compiladores (que estarán en **\$QTENV/msys64**) donde instalará los fuentes descargados con git (**\$QTENV/fluidsynth-2.4.6**) y las librerías ya compiladas (en el caso de 64 bits en **\$QTENV/fluidsynth-64-static** la estática y en **\$QTENV/fluidsynth-64** la compartida. La integración en Midieditor, debe hacerse a mano, llevando a **midieditor\lib64\windows** los ficheros creados **libfluidsynth-3.dll** y **libfluidsynth-3.dll.a** en el caso de librería compartida y a **midieditor\lib64\windows_static** el **libfluidsynth-3.a** de la librería estática. También es conveniente copiar los ficheros de cabecera .h de fluidsynth en **midieditor\fluidsynth**, exceptuando **fluidsynth.h**, que va en **midieditor\src\fluid** .

Mientras estemos en la versión de Fluidsynth 2.4.6 no es necesario hacer nada de esto, ni siquiera compilar, pues ya suministro yo las librerías compiladas. Pero las herramientas están ahí en caso de necesidad o de actualización (si abres con un editor de texto **build-fluidsynth64bits.sh** al inicio verás **FSVERSION="2.4.6"** . Modificando ese número de versión podrás obtener las actualizaciones correspondientes.)

Uso de la herramienta build_fluidsynth64bits.bat

Al ejecutar **build_fluidsynth64bits.bat** verás una pantalla tal que así:

```
C:\WINDOWS\system32\cmd.exe
Building with MSYS64 fluidsynth library
Select an option:
1 - static build
2 - shared build
0 - Exit
Press 1, 2 or 0:
```

1 - static build

Pulsa 1 para crear la versión estática

2 - shared build

Pulsa 2 para crear la versión compartidas

0 – Exit

Pulsa 0 para salir de la aplicación.

Si pulsamos 2 nos mostrará esta pantalla:

```
C:\WINDOWS\system32\cmd.exe
Building with MSYS64 fluidsynth library
Select an option for shared build:
1 - configure
2 - build
3 - install
4 - clean
0 - Exit
Press 1, 2, 3, 4 or 0: _
```

1 – configure

Descarga y configura el código fuente de fluidsynth 2.4.6

2 – build

Compila la librería y la aplicación fluidsynth.exe (en este caso, de tipo compartido por lo que usara DLL)

3 – install

Instala el resultado de la compilación en el directorio **\$QTENV/fluidsynth-64** en este caso.

4 – clean

Borra el directorio **\$QTENV/fluidsynth-build-64** con el resultado de la compilación (una vez instalado, es mejor borrar ficheros objeto que ocupan espacio en balde)

0 – Exit

Sale de la aplicación.

La compilación de Midieditor Custom

Una vez tenemos todo lo necesario para poder compilar Midieditor Custom, falta saber los pasos necesarios para poder hacerlo. El fichero **midieditor.pro** (midieditor/midieditor.pro) es el que utiliza la aplicación qmake para configurar y construir la aplicación. Como argumentos a qmake se le pasan tanto la versión a construir como si queremos realizar la compilación usando librerías estáticas de esta forma:

```
qmake.exe ... "DEFINES += __ARCH64__" "STATIC_BUILD = 1"... // 64 bits static
```

```
qmake.exe ... "DEFINES += __ARCH32__" "STATIC_BUILD = 1"... // 32 bits static
```

evidentemente, para compilar en 32 bits por ejemplo, tendrás que tener los compiladores y librerías necesarias.

Sobre midieditor.pro

Aparte de esos argumentos externos que configuran la arquitectura y la compilación estática, existen otros que internamente, modifican la construcción del código:

version de Midieditor Custom:

MIDIEDITOR_RELEASE_DATE="2025/05"

MIDIEDITOR_RELEASE_VERSION_ID= "0"

MIDIEDITOR_RELEASE_VERSION_STRING="CUSTOM12LEs"

Usar Fluidsynth estático/o compartido (# sirve para comentar la línea e ignorarla):

#WITH_FLUIDSYNTH=USE_FLUIDSYNTH_STATIC # static

WITH_FLUIDSYNTH=USE_FLUIDSYNTH # dynamic

nota: si se comentan esas dos líneas (añadir # delante) el resultado será una compilación sin soporte Fluidsynth.

Variables de entorno.

QTENV → (por ejemplo: QTENV=[F:/Qt](#)) necesaria para las herramientas que trabajan usando el entorno MSYS2

Path+=F:\Qt\msys64\usr\bin → apuntando a las herramientas de MSYS como sh

INSTALLJAMMER=F:\Qt\installjammer\installjammer → ruta (MSDOS) a installjammer. Esta aplicación se utiliza para crear un instalador para Midieditor Custom con todo lo necesario (DLLs, recursos...)

descargar installjammer:

<https://github.com/damoncourtney/installjammer>

<https://sourceforge.net/projects/installjammer/>

Para MSYS2 editar el fichero installjammer y cambiar:

PLATFORM="" por PLATFORM="Windows"

Compilar con qtbuild_msys_win32bits-static.bat / qtbuild_msys_win64bits-static.bat

En **midieditor\building**, aparte de las herramientas para compilar **Fluidsynth**, también tenemos estas dos para compilar **Midieditor Custom** sin depender de **Qt Creator**. Si lanzamos **qtbuild_msys_win64bits-static.bat** veremos una pantalla así:



```
C:\WINDOWS\system32\cmd.exe
QTENV=F:\Qt

Building with MSYS

Select an option:

1 - Configure
2 - Build
3 - Run
4 - Install
5 - Clean
0 - Exit

Press 1, 2, 3, 4, 5 or 0: _
```

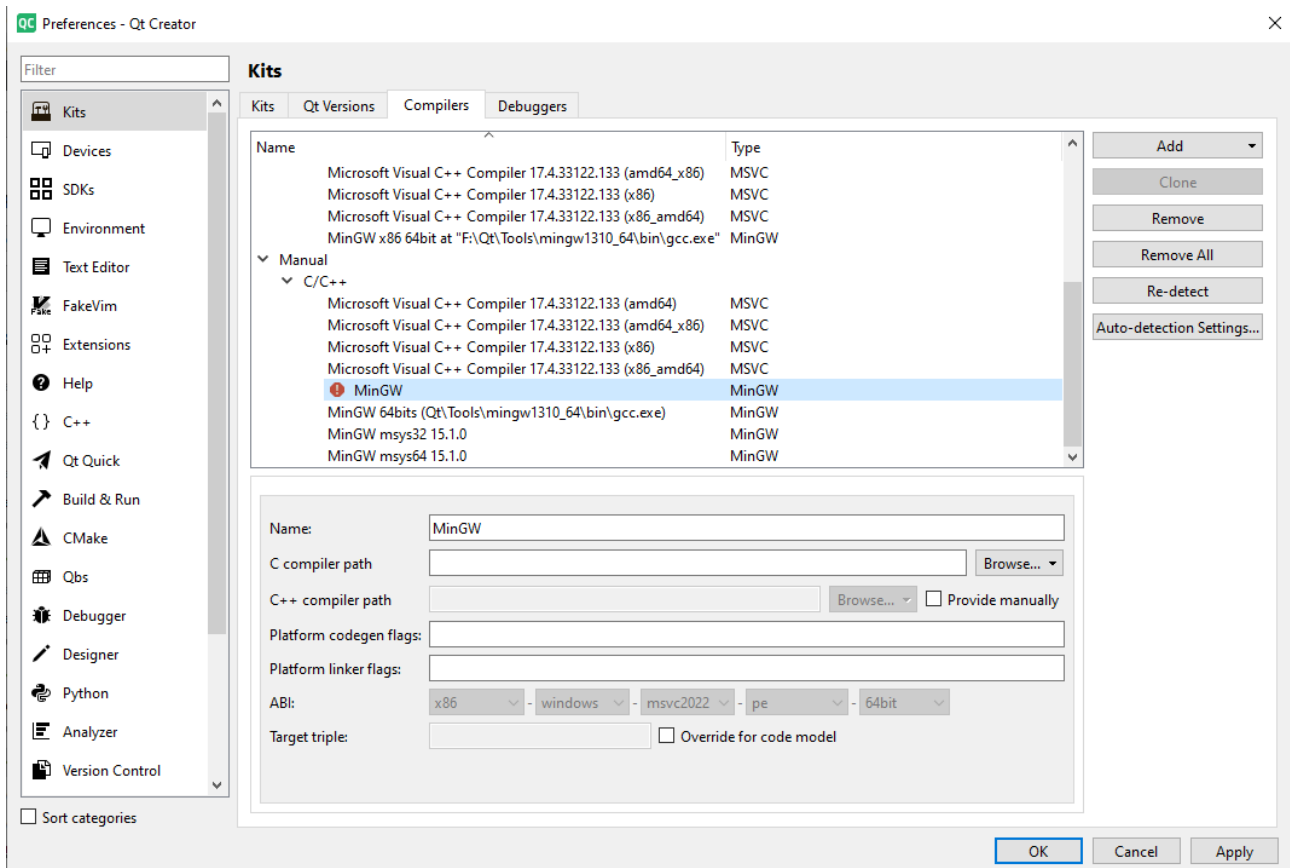
- 1 – Configure → configura en este caso, para aplicación de 64 bits, con librerías estáticas
- 2 – Build → Compila Midieditor.
- 3 – Run → Ejecuta Midieditor (una vez compilado, claro)
- 4 – Install → Empaqueta y crea un instalador para Midieditor (se debe tener midieditor.exe construido)
- 5 – Clean → Borra el código construido para una nueva compilación.
- 0 – Exit → Salir

La compilación se realiza en **midieditor\build\build-Midieditor-64bits-static** y dentro de ese directorio tendremos el ejecutable en **\bin**, **installjammer** trabajará sobre **\MidiEditor-win64** y dejará el instalador ya empaquetado en **\install**

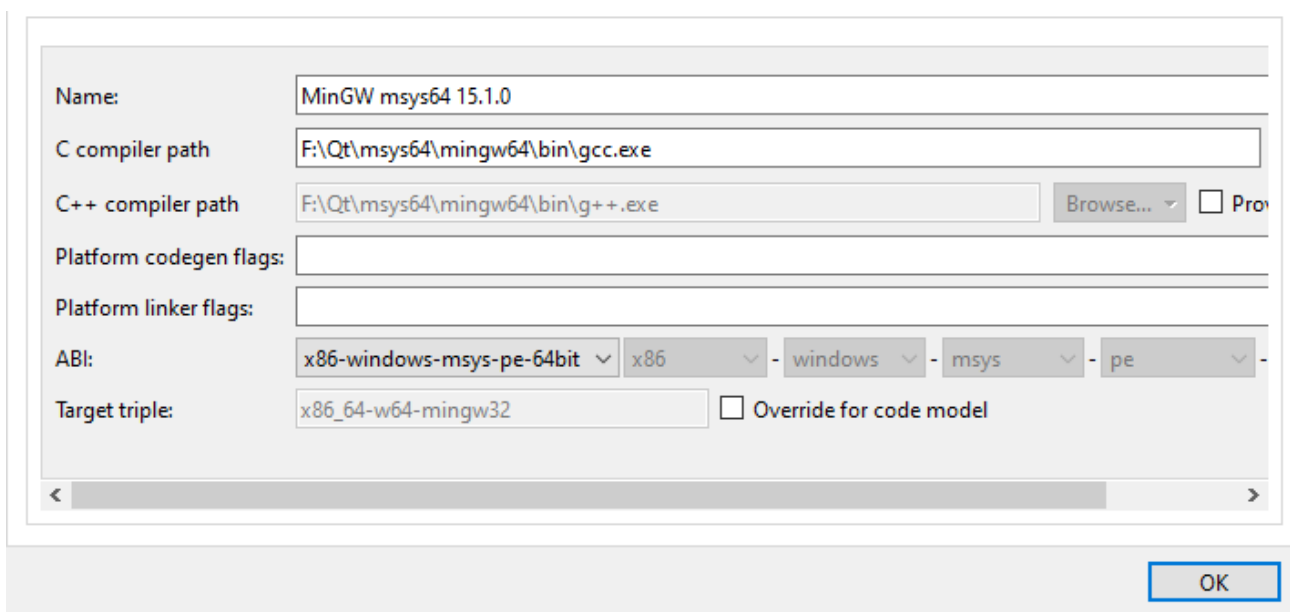
Compilar desde Qt Creator

Si Qt Creator está instalado, haciendo doble click sobre **midieditor.pro** abrirá el proyecto. Aquí deberíamos lo primero, crear el kit que vamos a usar para compilar. Para ello, en la barra lateral pulsamos el icono de **“Projects”** y luego sobre **“Manage kits”**.

En la pestaña **“Compilers”** pulsamos el botón **“Add”** y seleccionamos **“MinGW”**. Veremos algo así:



“Name” debería tener un nombre suficientemente descriptivo y lo que hay que hacer ahora, es en “C compiler path”, buscar gcc.exe. Ejemplo de mi caso para el compilador de 64 bits:



Ahora lo siguiente es pulsar la pestaña “Qt Versions” y luego en el botón “Add”. Nos

abrirá una ventana para buscar el fichero qmake.exe, que si has creado las librerías/herramientas Qt estáticas con las herramientas que describo aquí, deberían estar en (sumiendo [F:\Qt](#) como QTENV) **F:\Qt\6.9.0\mingw_msys32_static\bin\qmake.exe** para **32 bits** y en **F:\Qt\6.9.0\mingw_msys64_static\bin\qmake.exe** para **64 bits**. Detalle para 64 bits:

The screenshot shows a Qt Kit configuration window. The 'Name' field is set to 'Qt %\{Qt:Version\} (mingw_msys64_static)'. The 'qmake path' is 'F:\Qt\6.9.0\mingw_msys64_static\bin\qmake.exe'. The 'Qt version' is '6.9.0 for Desktop'. The 'Register documentation' dropdown is set to 'Highest Version Only'. At the bottom right are 'OK' and 'Cancel' buttons.

Una vez que ya tenemos compiladores y las herramientas/librerías de Qt, queda crear el kit. Para ello pulsar en la pestaña “**Kits**” y luego en el botón “**Add**”. Como referencia, dejo la configuración de mi kit para 64 bits estático.

En “**Name**” → Desktop Qt %\{Qt:Version\} MinGW 15.1.0 64-bit static

En “**Compiler**” → MinGW msys64 15.1.0

En “**Qt Version**” → Qt 6.9.0 (mingw_msys64_static)

y “**OK**”

The screenshot shows a detailed Qt Kit configuration window. The 'Name' field is 'Desktop Qt %\{Qt:Version\} MinGW 15.1.0 64-bit static'. The 'File system name' is empty. The 'Build device' and 'Run device' are both set to 'Desktop' with 'Local PC (default for Desktop)' as the device. The 'Compiler' is 'C/C++: MinGW msys64 15.1.0'. The 'Environment' section has 'Force UTF-8 MSVC output' unchecked and buttons for 'Edit Build Environment...' and 'Edit Run Environment...'. The 'Debugger' is 'Auto-detected CDB at C:\Program Files\Windows Kits\10\Debuggers\x64\cdb.exe'. The 'Sysroot' is empty. The 'Qt version' is 'Qt 6.9.0 (mingw_msys64_static)' and 'Mkspec' is empty. The 'Qbs Profile Additions' section is empty. The 'CMake Tool' is 'CMake 3.30.5 (Qt) (Default)'. The 'CMake generator' is 'Ninja'. The 'CMake Configuration' is '-DQT_QMAKE_EXECUTABLE:FILEPATH= %\{Qt:qmakeExecutable\} -DCMAKE_PREFIX_PATH:PATH= %\{Qt:QT_INSTALL_P...'. At the bottom right are 'OK', 'Cancel', and 'Apply' buttons.

Nota: Al añadir la Qt Versión, con otras compilaciones me salió un error relacionado con la ausencia de **qt.conf** (directorio **\bin**). Ese archivo tiene el siguiente contenido:

[Paths]

Documentation=../Docs/Qt-6.9.0

Examples=../Examples/Qt-6.9.0

Prefix=..

Ahora que ya tenemos creado nuestro kit, falta añadirlo en Midieditor. Para ello en **“Projects”**

→ **“Build & Run”** añadimos nuestro kit que se mostrará translucido y con el signo **“+”** para añadir.

Una vez hecho pulsamos **“Build”** y a nuestra derecha, arriba veremos **“Build Settings”**. Eliminar de **“Edit build configuration”** **“Debug”** y **“Profile”** pulsando sobre el botón **“Remove”** (solo dejamos **“Release”**)

y luego en **“Build Steps”** pulsamos en **“Details”** y añadimos en **“Additional arguments”**:

```
"DEFINES += __ARCH64__" "STATIC_BUILD = 1"
```

Y en **“Build Environment”** pulsamos **“Details”** buscamos **“Path”** hacemos click y continuación **“Append Path”** y buscamos en MSYS2 el directorio **msys64\usr\bin**, aunque también se puede añadir a la derecha directamente: **Path+=F:\Qt\msys64\usr\bin** (en mi caso, que lo tengo instalado ahí) para que encuentre **sh.exe, cmake...**

Aquí podeis añadir variables de entorno sin necesidad de añadirlas en el sistema. Por ejemplo la de **INSTALJAMMER**

Ahora pulsamos **“Run”** a la izquierda y veremos **“Run Settings”** arriba a la derecha. En

“Deployment” → **“Method:”** hacemos click en **“Add”** y añadimos **“Deploy Configuration”**.

Hacemos click en **“Deploy Configuration”** y seleccionamos **“Deploy Configuration2”**. Y ahora sobre el botón **“Rename...”** renombramos como **“Deploy Install”**.

Seguidamente pulsamos sobre **“Add Deploy Step”** y seleccionamos **“Make”**. En **“Make Arguments:”** añadimos: **packing**

Y ya tenemos todo preparado para poder compilar desde Qt Creator.