

[back](#)

Index of contents

- [the MinGW + MSYS environment](#)
- [preparing to use PKG-CONFIG](#)
- [Step 1\) building libiconv](#)
- [Step 2\) building libz](#)
- [Step 3\) building libjpeg](#)
- [Step 4\) building libpng](#)
- [Step 5\) building libtiff](#)
- [Step 6\) building libproj](#)
- [Step 7\) building libgeotiff](#)
- [Step 8\) building libgeos](#)
- [Step 9\) building libexpat](#)
- [Step 10\) building libspatialite](#)
- [Step 11\) building spatialite-tools](#)
- [Step 12\) building wxWidgets MSW](#)
- [Step 13\) building libfreetype](#)
- [Step 14\) building libfontconfig](#)
- [Step 15\) building libpixmap](#)
- [Step 16\) building libcairo](#)
- [Step 17\) building libgailgraphics](#)
- [Step 18\) building spatialite_gui](#)
- [Step 19\) building OpenSSL](#)
- [Step 20\) building libcurl](#)

the MinGW + MSYS environment

MinGW is an *open source* C/C++ compiler based on the popular **gcc**; **MSYS** is a command shell supporting a minimalistic Linux-like environment on Windows.

Using both them you can build standard open source software [*originally developed for Linux*] under Windows as well, simply using the *classic* build tool-chain, as in:

```
./configure  
make  
make install
```

Quite obviously, Windows isn't exactly the same as Linux, so you cannot expect to get always an easy and painless build.

May well be you have to adapt something here and there in order to get a successful build under Windows. The following notes are aimed exactly to this goal: allow you to avoid wasting your time while fighting against oddities that can quite easily solved.

We'll suppose you are using the latest **MinGW** and **MSYS**

So, if you have installed anything using the default settings [*the wisest thing to do*], now you have the following path correspondence:

C:\MinGW\msys\1.0\local will be mapped [*in the MSYS own perspective*] as: **/usr/local**

Accordingly, this one will be the standard home for any software you'll then build and install.

preparing to use PKG-CONFIG

pkg-config is a well known package configuration manager; it's widely used by many *open source* packages. Unhappily **pkg-config** doesn't come already installed once you've installed **MinGW** and **MSYS**. And, to make things worst, installing **pkg-config** on Windows is quite difficult and not at all straightforward.

First of all, you must download [pkg-config.exe](#) from **GTK+ for Windows**. Then you can simply *unzip* this downloaded zip-file, and then copy the **pkg-config.exe** executable into **/MinGW/bin**

That's not enough: this executable depends on the **GLib DLL**. So you must download [GLib DLL](#) too, always from **GTK+ for Windows**. Once again, you have to *unzip* this downloaded zip-file, and then copy **glib-2.0-0.dll** into **/MinGW/bin**

You have not yet finished: **pkg-config** still has an unresolved DLL dependency. But this time simply performing a trivial copy will suffice. So you must now open an MSYS shell:

```
cd C:/MinGW/bin
cp libintl-8.dll intl.dll
```

And this time that's really all: now you have **pkg-config** properly installed and ready to work.

Step 1) building libiconv

libiconv is the standard **GNU** library supporting *locale charsets*.

Required by: **libspatialite**, **spatialite-tools**

Building under Windows is not too much difficult.

- download [libiconv-1.13.1.tar.gz](#)
- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd libiconv-1.13.1
./configure
make
make install-strip
```

Anyway, this will simply build and install the DLL: a further step is required in order to get the *static library* as well.

```
make distclean
./configure --disable-shared
make
make install-strip
```

Now you've built and installed both the *static library* and the DLL.

However the above process has installed badly misconfigured **libcharset.la** and **libiconv.la** files (which are required to build other libraries in the following steps).

So in order to get a properly configured **libiconv** you have to accomplish a further operation:

- download [libiconv.la](#) and [libcharset.la](#)
- then copy both files: `cp libiconv.la libcharset.la /usr/local/lib`

Step 2) building libz

libz is a popular library implementing *Deflate*, i.e. the compression algorithm used by **gzip** and **Zip**.

Depends on: **nothing**

Required by: **libpng**, **libtiff**, ...

Building under Windows is quite easy, but requires to pay some attention.

- download the latest sources: [zlib125.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd zlib-1.2.5
make -f win32/Makefile.gcc
```

Now you simply have to manually copy the following files:

```
cp zlib1.dll /usr/local/bin
cp zconf.h zlib.h /usr/local/include
cp libz.a /usr/local/lib
cp libz.dll.a /usr/local/lib/libz.dll.a
```

All this will build and install both the *static library* and the DLL as well.

Anyway this process will not generate the **libz.la** file (which is required to build **libtiff** in one of the following steps).

So in order to get a fully installed **libz** you have to accomplish a further operation:

- download [libz.la](#)
- and then copy this file: `cp libz.la /usr/local/lib`

Step 3) building libjpeg

libjpeg is a popular library supporting the **JPEG** image compression.

Depends on: **nothing**

Required by: **libtiff**, **libgaiagraphics**

Important notice: you can now choose between two alternative implementations:

- **libjpeg** is the standard, plain library
- **libjpeg-turbo** is a new library, that fully takes profit from the most recent Intel/AMD CPUs
If you are planning to deploy your software on such platforms, then using *libjpeg-turbo* can ensure a **200%** performance boost (and even more than this).
I strongly recommend using *libjpeg-turbo*: both libraries share the same identical API/ABI (they are absolutely inter-changeable), but *libjpeg-turbo* runs in an impressively faster mode.

Building the one or the other under Windows is absolutely a plain and easy task.

How-to-build libjpeg-turbo

Please note: the **NASM** assembler is absolutely required: if you don't have it already installed on your system, you can [download](#) and install now.

- download the latest sources: [libjpeg-turbo-1.1.1.tar.gz](#)
- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd libjpeg-turbo-1.1.1
./configure --prefix=/usr/local
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

How-to-build libjpeg

- download the latest sources: [jpegsrc8b.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd jpeg-8b
./configure
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

Step 4) building libpng

libpng is a popular library supporting the **PNG** image compression.

Depends on: **libz**

Required by: **libgaiagraphics**

Building under Windows is absolutely a plain and easy task.

- download the latest sources: [libpng-1.5.2.tar.gz](#)
- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd libpng-1.5.2
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure
make
make install-strip
```

Important notice: you have to properly set the shell environment in order to retrieve the already installed **libz**; this is the duty of the two above **export** directives.

This will build and install both the *static library* and the DLL as well.

Important notice #2: in order to get **./configure** support you must absolutely download the **.tar.gz**, because the **.zip** doesn't supports it.

Step 5) building libtiff

libtiff is a popular library supporting the **TIFF** image format.

Depends on: **libz**, **libjpeg**

Required by: **libgaiagraphics**

Building under Windows is absolutely a plain and easy task.

- download the latest sources: [tiff-3.9.5.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd tiff-3.9.5
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
```

```
./configure
make
make install-strip
```

Important notice: you have to properly set the shell environment in order to retrieve the already installed **libz**; this is the duty of the two above **export** directives.

This will build and install both the *static library* and the DLL as well.

Step 6) building libproj

libproj is a library supporting coordinate's transformation between different Reference Systems [**PROJ.4**]

Depends on: **nothing**

Required by: **libgeotiff**, **libspatialite**, **spatialite-tools**

Building under Windows is an easy task.

- download the latest sources: [proj-4.7.0.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd proj-4.7.0
./configure --without-mutex
make
make install-strip
```

Important notice: may well be you'll get the following fatal errors:

```

pj_mutex.c:167: error: redefinition of 'pj_acquire_lock'
pj_mutex.c:65: error: previous definition of 'pj_acquire_lock' was here
pj_mutex.c:181: error: redefinition of 'pj_release_lock'
pj_mutex.c:75: error: previous definition of 'pj_release_lock' was here
pj_mutex.c:192: error: redefinition of 'pj_cleanup_lock'
pj_mutex.c:82: error: previous definition of 'pj_cleanup_lock' was here
pj_mutex.c:206: error: redefinition of 'pj_init_lock'
pj_mutex.c:91: error: previous definition of 'pj_init_lock' was here
```

in such an evenience you have to edit the **-/src/pj_mutex.c** source as follows:

```

33c33
< #ifndef _WIN32
---
> #if defined (_WIN32) && !defined(__MINGW32__)
_____

40c40
< #ifndef _WIN32
---
> #if defined (_WIN32) && !defined(__MINGW32__)
```

Step 7) building libgeotiff

libgeotiff is a library supporting the **GeoTIFF** raster format

Depends on: **libtiff**, **libproj**

Required by: **libgaiagraphics**

Building under Windows is an easy task.

- download the latest sources: [libgeotiff130.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd libgeotiff-1.3.0
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure --enable-incode-epsg
make
make install-strip
```

Important notice: it doesn't seem possible to build as a DLL using MinGW + MSYS. AFAIK, there is no way to do such a thing.

So you have to manually apply the following patch to circumvent this issue. Edit the **/usr/local/lib/libgeotiff.la** file as follows:

```
10c10
< library_names="
---
> library_names='libgeotiff.a'
```

Step 8) building libgeos

libgeos is a library representing a **C++** porting of **JTS** [*Java Topology Suite*].

Depends on: **nothing**

Required by: **libspatialite**, **spatialite-tools**

This library really is an **huge and complex** piece of software; building on Windows is an incredibly time consuming task.

- download the latest sources: [geos-3.3.0.tar.bz2](#)
- uncompress this bzip2-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd geos-3.3.0
./configure
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

Step 9) building libexpat

libexpat is a well known standard library supporting **XML parsing**.

Depends on: **nothing**

Required by: **libfontconfig**, **spatialite-tools**, ...

Building under Windows really is a piece-of-cake.

- download the latest sources: [expat-2.0.1.tar.gz](#)

- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd expat-2.0.1
./configure
make
make install
```

This will build and install both the *static library* and the DLL as well.

Step 10) building libspatialite

libspatialite is the main core of **SpatiaLite**

Depends on: **libiconv**, **libproj**, **libgeos**

Required by: **spatialite-tools**, **librasterlite**

Building under Windows is an easy task.

- download the latest sources: [libspatialite-amalgamation-3.0.0-beta.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd libspatialite-amalgamation-3.0.0-beta
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure --target=mingw32
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

Step 11) building spatialite-tools

spatialite-tools the **SpatiaLite** *command-line* management tools

Depends on: **libiconv**, **libproj**, **libgeos**, **libspatialite**, **libexpat**

Building under Windows is an easy task.

- download the latest sources: [spatialite-tools-3.0.0-beta.zip](#)
- uncompress this zip-file
- then open an MSYS shell

First of all, you must check if you've already installed **pkg-config.exe**

If not, please read the above [instructions](#)

And now you must set the **PKG_CONFIG_PATH** as appropriate:

```
export "PKG_CONFIG_PATH=/usr/local/lib/pkgconfig"
```

After this you are now ready to build as usual:

```
cd spatialite-tools-3.0.0-beta
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure --target=mingw32
make
make install-strip
```

Please note: following the above method you'll get *dynamically linked* tools [i.e. depending on DLLs]. If you wish instead to build *statically linked* tools [i.e. self contained, not depending on DLLs], now type:

```
mkdir static_bin
make -f Makefile-static-MinGW
cp static_bin/* /usr/local/bin
```

Step 12) building wxWidgets MSW

wxWidgets is a popular *widgets* library, supporting GUI in a cross-platform fashion; **MSW** is the specific porting supporting Windows.

Depends on: **nothing**

Required by: **spatialite-gui, spatialite-gis**

This library really is an huge and complex piece of software; building on Windows is an incredibly time consuming task, but is quite plain and easy.

- download the latest sources: [wxMSW-2.8.12.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd wxMSW-2.8.12
mkdir msw_build
cd msw_build
../configure --disable-shared --disable-debug \
  --disable-threads --enable-monolithic --enable-unicode \
  --without-libjpeg --without-libpng --without-zlib\
  --without-libtiff --without-expat --without-regex
```

Please note: the wxMSW `./configure` is highly configurable: you must apply exactly the above settings. Anyway, when `./configure` stops, it's a good practice to check if the final report looks exactly like this:

```
Configured wxWidgets 2.8.12 for `i686-pc-mingw32'

Which GUI toolkit should wxWidgets use?          msw
Should wxWidgets be compiled into single library? yes
Should wxWidgets be compiled in debug mode?      no
Should wxWidgets be linked as a shared library?  no
Should wxWidgets be compiled in Unicode mode?    yes
What level of wxWidgets compatibility should be enabled?
wxWidgets 2.4                                     no
wxWidgets 2.6                                     yes
Which libraries should wxWidgets use?
jpeg                                              no
png                                              no
regex                                           no
tiff                                             no
zlib                                             no
odbc                                             no
expat                                           no
libmspack                                       no
sdl                                             no
```

now, when `./configure` stops, you have to continue as usual:

```
make
make install-strip
```


Step 13) building freetype

libfreetype is a standard library supporting **TrueType** fonts.

Depends on: **nothing**

Required by: **libcairo**, ...

Building under Windows is an easy task.

- download the latest sources: freetype-2.4.4.tar.gz
- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd freetype-2.4.4
./configure
make
make install
```

This will build and install both the *static library* and the DLL as well.

Step 14) building libfontconfig

libfontconfig is a standard library supporting **font customization** and configuration.

Depends on: **libexpat**, **libfreetype**, **libiconv**

Required by: **libcairo**, ...

Building under Windows is an easy task.

- download the latest sources: fontconfig-2.8.0.tar.gz
- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd fontconfig-2.8.0
export "CFLAGS=-I/usr/local/include -DLIBICONV_STATIC"
export "LDFLAGS=-L/usr/local/lib"
./configure --disable-docs
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

Step 15) building libpixman

libpixman is the standard library implementing **pixel manipulation** for Cairo.

Depends on: **nothing**

Required by: **libcairo**, ...

Building under Windows is an easy task.

First of all, you must check if you've already installed **pkg-config.exe**

If not, please read the above [instructions](#)

And now you must set the **PKG_CONFIG_PATH** as appropriate:

```
export "PKG_CONFIG_PATH=/usr/local/lib/pkgconfig"
```

All right, your system configuration is ready to build *fontconfig*, so you can now:

- download the latest sources: pixman-0.20.2.tar.gz
- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd pixman-0.20.2
./configure
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

Step 16) building libcairo

libcairo is a very popular **graphics** library.

Depends on: **libpixman**, **libfreetype**, **libfontconfig**, **libpng**

Required by: **libgaiagraphics**, ...

Building under Windows is a little bit harder than usual.

First of all, you must check if you've already installed **pkg-config.exe**

If not, please read the above [instructions](#)

And now you must set the **PKG_CONFIG_PATH** as appropriate:

```
export "PKG_CONFIG_PATH=/usr/local/lib/pkgconfig"
```

All right, your system configuration is ready to build *libcairo*, so you can now:

- download the latest sources: cairo-1.10.2.tar.gz
- uncompress this gzipped-file
- then untar the *tarball*
- and finally open an MSYS shell

```
cd cairo-1.10.2
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure --disable-pthread
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

Step 17) building libgaiagraphics

libgaiagraphics is a common utility library supporting **graphics rendering**

Depends on: **libjpeg**, **libpng**, **libtiff**, **libgeotiff**, **libcairo**

Required by: **spatialite-gui** [*next-to-come* releases of **librasterlite** and **spatialite-gis**]

Building under Windows is an easy task.

First of all, you must check if you've already installed **pkg-config.exe**

If not, please read the above [instructions](#)

And now you must set the **PKG_CONFIG_PATH** as appropriate:

```
export "PKG_CONFIG_PATH=/usr/local/lib/pkgconfig"
```

All right, your system configuration is ready to build *libgaiagraphics*, so you can now:

- download the latest sources: [libgaiagraphics-0.4.zip](#)
- uncompress this zip-file
- then open an MSYS shell

```
cd libgaiagraphics-0.4
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure --target=mingw32
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

Step 18) building spatialite_gui

spatialite_gui the **Spatialite** *GUI* user-friendly tool

Depends on: **libspatialite**, **wxWidgets**, **libgaiagraphics**

Building under Windows is an easy task.

- download the latest sources: [spatialite_gui-1.5.0-beta.zip](#)
- uncompress this zip-file
- then open an MSYS shell

First of all, you must check if you've already installed **pkg-config.exe**

If not, please read the above [instructions](#)

And now you must set the **PKG_CONFIG_PATH** as appropriate:

```
export "PKG_CONFIG_PATH=/usr/local/lib/pkgconfig"
```

After this you are now ready to build as usual:

```
cd spatialite_gui-1.5.0-beta
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure
make
make install-strip
```

Please note: following the above method you'll get a *dynamically linked* GUI tool [i.e. depending on DLLs]. If you wish instead to build a *statically linked* GUI tool [i.e. self contained, not depending on DLLs], now type:

```
mkdir static_bin
make -f Makefile-static-MinGW
cp static_bin/* /usr/local/bin
```

Step 19) building OpenSSL

OpenSSL is a well known standard library supporting **SSL**, i.e. the **encrypted HTTPS** web protocol.

Depends on: **nothing**

Required by: **libcurl**

Building under Windows is a little bit difficult, and requires to pay close attention.
The **configure** script isn't at all a standard one: please read carefully the following instructions.

- download the latest sources: [openssl-1.0.0d.tar.gz](#)
- **Important notice:** you cannot use tools such as **7z** to untar the *tarball*: this will cause fatal errors during compilation (*broken links*).

You absolutely have to run all the following commands from the MSYS shell.

```
tar zxvf openssl-1.0.0d.tar.gz
cd openssl-1.0.0d
./Configure mingw --prefix=/usr/local shared
make
make install
```

This will build and install both the *static libraries* and the DLLs as well.

Step 20) building libcurl

libcurl is a well known library supporting **URLs** (networking, web protocols)

Depends on: **libz**, **OpenSSL**

Required by: **?**

Building under Windows is an easy task.

- download the latest sources: [curl-7.21.7.zip](#)
- uncompress this zip-file
- then open an MSYS shell

First of all, you must check if you've already installed **pkg-config.exe**

If not, please read the above [instructions](#)

And now you must set the **PKG_CONFIG_PATH** as appropriate:

```
export "PKG_CONFIG_PATH=/usr/local/lib/pkgconfig"
```

After this you are now ready to build as usual:

```
cd curl-7.21.7
export "CFLAGS=-I/usr/local/include"
export "LDFLAGS=-L/usr/local/lib"
./configure
make
make install-strip
```

This will build and install both the *static library* and the DLL as well.

[back](#)