**Installation einer vollständigen Qt5/MinGW/MSYS-Arbeitsumgebung für Open-E-Government**

# Vorab zu installierende Anwendungen und Pakete

## Microsoft Windows SDK

<http://www.microsoft.com/en-us/download/details.aspx?id=8442>  
Microsoft Windows SDK for Windows 7 and .NET Framework 4 (ISO)

Needed for midl (ActiveX applications), signtool (all appliciations), guidgen/uuidgen (only development). If missing cl.exe (C preprocessor).

## Microsoft DirectX SDK

Die Datei stammt vom Juni 2010, wird aber benötigt, wenn Qt selbst kompiliert werden soll. Und auch nur dann. Also ggf. weglassen.

## MinGW Builds

Dieses Projekt bietet fertige Pakete des mingw-w64-Projekts mit MinGW-Compilern an, welche Dual-Target-Code erzeugen (also 32 und 64 Bit) und auf 32 und 64-Bit Windows laufen und CRT und die WinAPI benutzen. Sie haben nun die Wahl: Entweder das Qt-mingw nehmen, welches unter 32-Bit Windows läuft und Code für 32 und 64 Bit erzeugen kann. Damit wird es aber ggf. komplizierter, OSS-Tools (configure & make & make install) zu bauen, da verschiedene Pakete noch nicht soweit sind. Oder Sie installieren sich zwei identische mingw-Tool-Chains parallel (in verschiedenen Verzeichnissen). Diese können Sie benutzen, ohne die Namen der Tools in den Skripten ändern zu müssen. Das funktioniert natürlich nur, wenn Sie 64-Bit Windows laufen haben. Bisher nutzte ich das bei Qt mitgelieferte mingw32 und baute alles damit, achtete aber gleichzeitig darauf, dass das Bauen mit mingw64 auch funktioniert, also dass bspw. überall die richtigen Verzeichnisse verwendet werden.

Da Qt/MinGW ohne SSL-Bibliotheken ausgeliefert wurde und auch andere Kleinigkeiten (D-Bus) nicht so ohne weiteres aktivierbar waren, benutze ich derzeit die im mingw-w64-Projekt enthaltenen Kompiler (die mingwbuilds-Kompiler sind dorthin umgezogen) und die im mingwbuilds-veröffentlichten Qt-Bibliotheken. Später werde ich alles selbst übersetzen, einige der OSS-Pakete sind es jetzt auch schon.

Als Kompiler laden Sie sich die aktuelle Version mit posix-Threads-Implementierung herunter. Prinzipiell sollten die win32-Threads gleichwertig oder gar schneller funktionieren, aber posix ist derzeit auch die Toolchain, die für Qt genutzt wird. Man könnte irgendwann aber sicher die win32-Threads-Variante austesten – wir arbeiten schließlich unter Windows. Aufpassen: Die mingw-w64-Kompiler sind jeweils nur für 32 und 64 Bit verwendbar. Ein Paket kann nicht für beide Ziele genutzt werden, da dann Bibliotheken nicht gefunden werden. Ich habe das aber nicht genauer untersucht, sondern einfach für 32 und 64 Bit das jeweils andere Verzeichnis einkonfiguriert.

<http://sourceforge.net/projects/mingwbuilds/>

Cross-Compiler für Windows-Entwicklung unter Linux gibt es bspw. hier:

<http://sourceforge.net/projects/mingw-w64/files/>

mingw-w64-bin\_i686-linux\_20121031.tar.bz2  
mingw-w64-bin\_x86\_64-linux\_20121031.tar.bz2

Beide Verzeichnisse bspw. nach /opt entpacken und Sie können 32- und 64-Bit-Executables für Windows bauen.

## Fertiges Qt

Für die Zeit, bis Sie eine eigene Qt-Version übersetzt haben, sollten Sie einen bereits übersetzten Qt-Build einsetzen. Grundsätzlich sind folgende Alternativen möglich:

### Qt von Digia (offizielle Komplettversion mit MinGW/gcc)

Hier gibt es eine offizielle Release und eine inoffizielle Fassung, welche mit dem Build-Datum gekennzeichnet ist.

Version: 5.2.0  
Datum: 11.12.2013  
<http://download.qt-project.org/official_releases/qt/5.2/5.2.0/>

qt-windows-opensource-5.2.0-mingw48\_opengl-x86-offline.exe

Die einfachste Variante ist es, das offizielle Komplettpaket von Qt zu installieren. Wählen Sie die mingw-Variante. Die Visual Studio Builds sind vielleicht sogar effizienter oder schneller (nicht getestet), jedoch gebe ich dafür keine Hilfe, da das hier ein OSS-Projekt ist, welches nicht zuletzt aus Kosten- und Abhängigkeitsgründen auch nur mit solcher Software gebaut werden können soll.

Seit Anfang 2013 steht netterweise wieder eine aktuelle Version bereit, nachdem vorher für lange Zeit eine Version vom Februar 2010 die einzige vorkompilierte Version war. Sie installieren sich damit die Entwicklungsumgebung, eine funktionsfähige MinGW-Installation nebst aktuellem gcc (der gleiche wie bereits anderweitig installiert) und auch die Dokumentation und die Beispiele. Der Compiler ist nur in einer Minimalversion dabei und prinzipiell zwar in der Lage, 64-Bit-Code auch auf 32er Maschinen zu erstellen, jedoch wird es schwierig mit den beiliegenden autotools und configure-Skripten vieler OSS-Pakete, da diese bspw. testen, ob der Compiler funktionsfähigen Code erzeugen kann (was bspw. mit dem ‑m64-Flag problemlos klappt) und dann aussteigen, wenn das 64-Bit-Executable nicht auf dem 32-Bit-Rechner läuft. Die Angabe von --host wiederum mögen einige andere Pakete nicht, so wird bei libiconv bspw. dann keine Windows-DLL erstellt. Auch scheint es einfacher zu sein, getrennte Compiler für 32 und 64-Bit Entwicklung zu benutzen. Aus diesen Gründen wird der mit Qt mitgelieferte gcc im 64-Bit-Fall nicht verwendet. Aber wenn Sie schnell in der IDE ein Programm schreiben oder ein Beispiel übersetzen wollen, dann ist es einfacher, diesen gleich mit zu installieren und in solchen Fällen zu verwenden.

Der Compiler kann bei Bedarf noch um weitere Tools (SSL-Unterstützung) ergänzt werden:  
<http://origin.releases.qt-project.org/prebuilt/mingw_32/>

Sie sollten aber genau wissen, was Sie warum hinzu installieren.

Unfortunately the start menu entries are only available for the account which installed Qt. That means if you installed as a regular user and used “install as Administrator”, then the administrator account also gets the start menu entries. In such a case simply copy the "Qt 5.2.0" directory from the start menu of the administrator account and insert it into your start menu.

### Qt aus dem mingwbuilds-Projekt

Diese Builds stammen vom gleichen Projekt wie die gcc-Compiler, die auch für die offiziellen Builds genommen werden. Der Vorteil ist, dass diese bereits in 32 und 64 Bit existieren und auch als statische Pakete (so man diese denn benutzen möchte) vorliegen.

<http://sourceforge.net/projects/mingwbuilds/files/external-binary-packages/Qt-Builds/>

Dateien herunterladen und Verzeichnisse beispielsweise auf C:\ ablegen:

x64-Qt-5.2.0+QtCreator-3.0.0-(gcc-4.8.2-she).7z

x32-Qt-5.2.0+QtCreator-3.0.0-(gcc-4.8.2-dwarf).7z

Um die Binaries aus diesen Paketen benutzen zu können, musste bei den frühen Qt5-Versionen noch eine DLL jeweils im bin-Verzeichnis hinzugefügt werden: D3DCompiler\_43.dll.

<http://origin.releases.qt-project.org/prebuilt/>

D3DCompiler\_43-x64.dll.7z

D3DCompiler\_43-x86.dll.7z

Nun muss noch in Qt/bin/qt.conf das Basisverzeichnis von Qt an den aktuellen Pfad angepasst werden:

C:\Qt32-5.1.0\bin\qt.conf - Verzeichnis korrigieren!

C:\Qt64-5.1.0\bin\qt.conf - Verzeichnis korrigieren!

Das ist aber nicht so wichtig: Einmal gibt es den qtbinarypatcher, der das automatisch erledigen kann und zum Anderen ist im OEG-Projekt alles so eingerichtet, dass es trotzdem funktionieren sollte (eigene qt.conf als Fallback).

## MSYS

Das originale MSYS ist etwas schwierig zu installieren, da sehr viele Pakete heruntergeladen und eingespielt werden müssen. Hier gibt es alles in einem Paket. Da MSYS nur mit einem veralteten gcc übersetzt werden kann, gibt es leider keine 64-Bit-Version. Nähere Informationen finden Sie hier:

<http://sourceforge.net/apps/trac/mingw-w64/wiki/MSYS>

Die Dateien befinden sich hier:

<http://sourceforge.net/projects/mingwbuilds/files/external-binary-packages/>

Die Datei msys+7za+wget+svn+git+mercurial+cvs-rev13.7z herunterladen und extrahieren. Das Verzeichnis msys irgendwo in einem Laufwerk ganz weit an der Wurzel installieren. Also beispielsweise als D:\dev\msys.

Diese MSYS-Version enthält keine rxvt.exe. Das ist ein anderer Befehlszeileninterpreter als Alternative zu cmd.exe. Wenn Sie diese Datei benötigen, dann kopieren Sie sich eine solche noch hinzu. Beispielsweise aus einem anderen MSYS-Paket.

Richten Sie sich nun noch ein MSYS-Icon ein, indem Sie einen Link auf die msys.bat erstellen und diesem als Icon eines der im gleichen Verzeichnis befindlichen Icons zuweisen.

Wenn Ihr Administrator die TEMP-Verzeichnisse irgendwohin umgeleitet hat und Sie ihre temporären Dateien bspw. nicht im Netzwerk ablegen wollen (erzeugt bei configure-Aufrufen viele Fehler!) und die Variablen nicht System-global korrigieren dürfen, dann fügen Sie in der MSYS.BAT ganz oben die entsprechenden neuen Pfade ein:

SET TMP=C:\TEMP

SET TEMP=C:\TEMP

Die Datei /etc/fstab musste früher noch angepasst werden. Meine sah einst so aus:

e:/mingw32 /mingw32

e:/mingw64 /mingw64

e:/msys /msys

e:/qt32-5.0.1 /qt32

e:/qt64-5.0.1 /qt64

e:/ActiveState/perl /perl

Achtung: Pfadangaben in MSYS sehen so aus: /c/XYZ steht für c:\XYZ. Und bei der fstab noch ein wenig anders. Bei einem Tippfehler gehen ggf. alle Zielpfade nicht!

Sie können MSYS auch einfach nur nach C:\ entpacken. Die fstab-Anpassungen sind nur für wenige Pakete wirklich notwendig. Jetzt befinden sich die Pfade in OEG/env.sh. Diese Datei wird in MSYS gesourced und erleichtert das Bauen der OSS-Packages.

## WiX Toolset

Das WiX Toolset steht in der Version 3.7.1224.0 bereit. Es dient dazu, die im Verzeichnis deployment enthaltenen Skripte auszuführen, welche aus fertig kompilierten und im bin-Verzeichnis liegenden Anwendungen installierbare msi-Archive zusammenstellen, welche ein Nutzer nur noch anklicken und installieren muss. Setup-Dateien mit einer exe-Datei werden gar nicht angeboten, da das MSI-Format für Windows-Pakete gedacht ist und auf aktuellen Rechnern so auch tadellos funktioniert. Sie benötigen das Toolset eigentlich nur, wenn Sie eigene Anwendungen oder selbst kompilierte Binaries veröffentlichen wollen.

Homepage  
<http://wixtoolset.org/>

Download  
<http://wix.codeplex.com/releases/view/99514>

Sie benötigen des Weiteren ein installiertes .NET Framework 4. Nutzer von Windows XP SP3 können auf die Version 4.0 zurück greifen, während Nutzer eines aktuelleren Betriebssystems das jeweils aktuelle .NET Framework 4.x (derzeit 4.5) installieren sollten. Dieses installiert sich unter alten Windows-Versionen nur leider nicht mehr. Bei Windows 8 ist das aktuelle .NET Framework bereits Teil des Betriebssystems. Die komplette Installationsdatei heißt entweder dotnetfx45\_full\_x86\_x64.exe oder dotnetfx40\_full\_x86\_x64.exe und ist etwa 50 MB groß.

Microsoft .NET Framework 4.5  
<http://www.microsoft.com/de-de/download/details.aspx?id=30653>  
Das .NET Framework 4.5 wird auf dem Betriebssystem Windows XP SP3 nicht unterstützt.  
dotnetfx45\_full\_x86\_x64.exe

Microsoft .NET Framework 4.0  
<http://www.microsoft.com/de-de/download/details.aspx?id=17718>  
dotnetfx40\_full\_x86\_x64.exe

## Kleinere Tools

### Dependency Walker

Das Tool depends.exe (Version 2.2) dient dazu, die Abhängigkeiten einer EXE-Datei zu ermitteln. Damit können Sie bspw. sicherstellen, dass alle notwendigen Dateien in Ihren Paketen enthalten sind.

<http://www.dependencywalker.com/>

## Testen

### Kompilieren nach 32 und 64 Bit

Geben Sie dieses Miniprogramm ein:

#include <stdio.h>

int main()

{

printf("Hello World!\n");

return 0;

}

Übersetzen geht so:

/mingw32/bin/i686-w64-mingw32-gcc hello.c -o hello-w32.exe  
/mingw64/bin/x86\_64-w64-mingw32-gcc hello.c -o hello-w64.exe

Wenn Sie nur einen Compiler verwenden (bspw. den im Qt-SDK mitgelieferten), dann fügen Sie -m32 bzw. -m64 zu den Compiler-Flags hinzu. Die getrennten Compiler sind auch in ihren jeweiligen Pfaden als gcc ansprechbar! Voraussetzung ist, dass Sie ein 64-Bit Windows einsetzen!

# Übersetzen gängiger OSS-Pakete

Es ist normalerweise nicht notwendig, Pakete selbst zu übersetzen, da das OEG-Projekt Archive für alle wichtigen Pakete vorkompiliert jeweils für 32- und 64-Bit bereitstellt.

## Environment

Die Arbeitsumgebung benötigt einige Variablen, welche der Einfachheit halber per Funktion gesetzt werden. Dazu diesen Code einfach in der Bash sourcen:

. /i/Projekte/open-egov/env.sh

## Vorbereitungen

oegenv 32

oegmingwpath 32

mkdir ${WORKDIR}

Wenn etwas mit dem Entpacken nicht klappt, dann einfach mit 7-zip extrahieren. Beispielsweise .xz-Dateien, wenn der Entpacker noch nicht vorliegt (müsste aber im MSYS-Paket dabei sein).

## Kompilieren per Hand bzw. builder/Bauhof-Anwendung

### libiconv

cd ${WORKDIR}/libiconv-1.14-32

#configure --host=i686 --build=i686 --prefix=${BASEDIR} -–enable-shared

#mit Angabe von --host oder --build werden mit dem Qt5-MinGW keine DLLs erzeugt.

configure --prefix=${BASEDIR} -–enable-shared

Compile again with gettext (libintl.dll) installed. Libiconv lässt sonst beim Erstellen ./share/locale/\* weg. Achtung, beim make install wird Zugriff auf die Daten von libintl benötigt (lib/libintl.a usw.).

configure --prefix=${BASEDIR} --enable-relocatable --with-gnu-ld --with-libiconv-prefix=${BASEDIR}

Der Parameter --enable-relocatable führt zu diesem Fehler:

\*) /bin/sh ../libtool --mode=link gcc -m32 -pipe -m32 -march=i686 -mtune=i686

-pipe -O2 -mms-bitfields -fomit-frame-pointer `if test -n ':'; then : /open-egov

ernment/bin; fi` iconv.o ../srclib/libicrt.a /open-egovernment/lib/libiconv.la -

L/open-egovernment/lib -lintl iconv.res -o iconv.exe;; \

esac

libtool: link: gcc -m32 -pipe -m32 -march=i686 -mtune=i686 -pipe -O2 -mms-bitfie

lds -fomit-frame-pointer iconv.o iconv.res -o .libs/iconv.exe ../srclib/libicrt

.a /open-egovernment/lib/libiconv.dll.a -L/open-egovernment/lib -lintl -L/open-e

government/lib

iconv.o:iconv.c:(.text.startup+0x5e): undefined reference to `relocate'

../srclib/libicrt.a(progreloc.o):progreloc.c:(.text+0x156): undefined reference

to `compute\_curr\_prefix'

../srclib/libicrt.a(progreloc.o):progreloc.c:(.text+0x16c): undefined reference

to `set\_relocation\_prefix'

collect2.exe: error: ld returned 1 exit status

Makefile:113: recipe for target `install' failed

make[1]: \*\*\* [install] Error 1

make[1]: Leaving directory `/work/libiconv-1.14/src'

Makefile:52: recipe for target `install' failed

make: \*\*\* [install] Error 2

setup: libiconv, gettext  
configure --prefix=${BASEDIR} --with-gnu-ld --with-libiconv-prefix=${BASEDIR}  
make

Make install benötigt verschiedene libs während des install-Vorgangs (da wird dann erst richtig gelinkt) im install-Verzeichnis. Da wir so nur schwer an eine saubere Installation (nur die neuen Dateien) kommen können, wird mit DESTDIR einfach alles woanders hin kopiert. Dort entsteht dann noch ein --prefix-Verzeichnis.

Achtung: Im share/locale-Verzeichnis werden nur die Locales übersetzt und installiert, die gewünscht sind. Also …

### gettext

tar xzf gettext-0.18.2.tar.gz

cd gettext-0. 18.2

./configure --prefix=/nonexistent --enable-relocatable

configure --prefix=${BASEDIR} --enable-threads=windows --enable-relocatable --with-gnu-ld --with-libiconv-prefix=${BASEDIR}

NEEDS: libiconv

./configure --enable-relocatable --prefix=/nonexistent

make

make install DESTDIR=/tmp/inst$$

## Benötigte OSS-Pakete

Die folgenden Pakete sollten bei der Entwicklung installiert sein bzw. können beim Release eigener Anwendungen vorausgesetzt werden.

### gettext

X

### libiconv

X

### zlib

X

## Eine saubere Arbeitsumgebung einrichten

Die Vorgehensweise zum sauberen Erstellen aller Binärdateien sieht folgendermaßen aus:

* Erstellen aktueller Pakete für gettext, zlib, libiconv mit dem builder oder per Hand mit MSYS und den bash-Funktionen aus diesem Dokument.
* Komplettes Löschen aller (oder ggf. nur der binären) Inhalte aus lib/lib64/bin/bin64/plugins/plugins64.
* Verzeichnis mit SVN aktualisieren (erzeugt einige exe-Dateien in bin)
* Entpacken der Binaries von gettext, zlib, libiconv in den letzten Versionen aus data/builder/archives/32|64 in besagte Verzeichnisse. Wer Sprachunterstützung, Handbücher, Beispiele etc. benötigt, kann diese optional nach share entpacken, d.h. einfach die kompletten Archive über das Hauptverzeichnis entpacken, ansonsten reichen die Dateien aus bin und lib.
* Öffnen eines Qt-Prompts und entweder alles mit build.bat automatisch bauen lassen oder per Hand: Erst die in src/libraries, dann die Anwendungen und zum Schluss die Plugins. Bei den Anwendungen zuerst die components. Wenn es Probleme dabei gibt, weil makesources.exe aufgerufen werden soll und es das nicht gibt, dann direkt ins src-Verzeichnis wechseln und da qmake && mingw32-make aufrufen. Dann werden die vorhandenen Dateien benutzt, statt alle aus der XML-Datei neu zu erstellen.

## MySQL

URL: <http://dev.mysql.com/downloads/>

Von MySQL gibt es ein Installationspaket, welches alles nötige bereits enthält. Laden Sie bspw. Den „MySQL Installer 5.6.10“ (mysql-installer-community-5.6.10.1.msi) für Ihre Plattform (Windows, 32 Bit) herunter.

Installation von: mysql-5.1.44-win32.msi, ohne Instance Manager und Embedded Server.

Firewall-Port: 3306  
Default character set: UTF-8  
bin in PATH, Service MySQL installiert (kein Autostart)  
root-PW, kein remote Zugriff vom Netzwerk  
C:\Program Files\MySQL\MySQL Server 5.1\my.ini

NOTIZEN AB HIER

Noch Patches von MinGW aus früheren Qt-Releases ansehen.

mingwrt-3.15.2-mingw32-dll

w32api-3.13-mingw32-dev

gmp-4.2.4-mingw32-dll

mpfr-2.4.1-mingw32-dll

pthreads-w32-2.8.0-mingw32-dll

gdb and libexpat-1.dll

mingwrt-3.18

GCC-PREREQUISITES

- binutils 2.13 - installiert: 2.20-1

- mingw-runtime 3.13 - installiert: 3.17

- w32api 2.2 - installiert: 3.14

DLL prerequisites: GMP, libiconv, MPFR, support for all languages,

POSIX Threads for Win32 Runtime [REQUIRED]: pthreads-w32-2.8.0-mingw32-dll.tar.gz

Be aware that some archive extracters do not preserve read-only attributes of files. If you are installing the Ada component, please check that the files in the /lib/gcc/mingw32/4.2.1-dw2/adainclude and adalib directories are flagged as read-only. This attribute is necessary to prevent them from being deleted when using gnatclean to clean a project.

For more information, see: <http://gcc.gnu.org/gcc-4.4/>.

Die MSYS-Installation soll alles enthalten, was für ein configure-fähiges System benötigt wird. Es ist nicht geplant, gegen MSYS-Bibliotheken zu linken, d.h. solche Bibliotheken sollten nur in MinGW enthalten und natürlich auch für MinGW und nicht für MSYS kompiliert worden sein.

## MSYS-Update

TODO: ….

Install MSYS Core 1.0.14-1 archives. Untar them in C:\msys\1.0.

## MSYS Developer Tool Kit

Msys DTK contains: autotools, autoconf, automake, libtool, autogen, guile, cvs, openssl, openssh, inetutils and perl. I’ve also included the /usr/share/vim directory for the already included /bin/vim binary. I’ve only included the client tools of any package providing both server and client. The cvs, openssl, openssh and inetutils are provided for remote operations with SourceForge. Perl is provided because autoconf, automake, libtool, texi2html (to generate the documentation) require it. Guile is provided because autogen requires it and autogen is provided, just because it’s a nice tool.

Die Qt-Pfade in die ~/.profile (nicht .bashrc) o.ä. eintragen und ausprobieren:

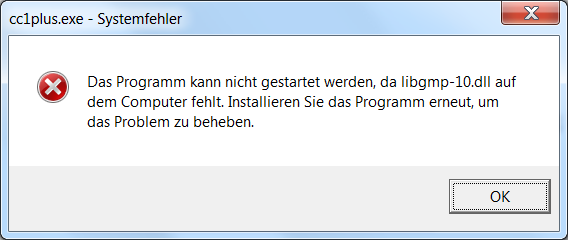
cd /qt/demos

cp –R books books2

rm –rf release debug tmp

qmake

make



Also nun noch gmp nachinstallieren (Verzeichnis: MinGW/nachinstalliert). Da gibt es 2 Versionen auf mingw.org, einmal für MinGW und einmal für MSYS. Da die MSYS-Version 4.3.1-1, und die für MinGW 5.0.1-1 ist, habe ich zuerst die MinGW-Variante ausprobiert. Außerdem ist MSYS nur für eigenständige Tools da, die den Build-Prozess unterstützen, da keine Abhängigkeiten entstehen sollen.

Benötigt wurden: libgmp-10.dll, libmpc-2.dll, libmpfr-1.dll

Nachinstalliert wurden:

libgmp-5.0.1-1-mingw32-dll-10.tar.lzma

libmpc-0.8.1-1-mingw32-dll-2.tar.lzma

libmpfr-2.4.1-1-mingw32-dll-1.tar.lzma

Interessant ist, dass nun auch die deutschen Umlaute in der bash funktionieren:

# gcc --version

Anpassungen

/home/gerrit/.bashrc

/home/gerrit/.profile

/etc/fstab

/etc/profile

/usr/open-egovernment/packs

tar --overwrite -xf msysCORE-1.0.11-20080826.tar

## Recommended updates for MSYS

MSYS Base System contains numerous updates for MSYS. It is recommended to install the following updates: Bash Coreutils m4

<http://downloads.sourceforge.net/mingw/m4-1.4.7-MSYS.tar.bz2>

<http://ftp.gnu.org/gnu/autoconf/> 2.65 MSYS: 2.63-1

<http://ftp.gnu.org/gnu/automake/> 1.11.1 MSYS: 1.11-1

<http://ftp.gnu.org/gnu/libtool/> 2.2.6b, 1.5.26 MSYS: 2.2.7a-1

If make seems to be stuck in an infinite loop, [csmake](http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=24963&release_id=46827) should probably be used instead. csmake is part of MSYS 1.0.11, installed as "make"

As all the libraries will be installed in /mingw, PKG\_CONFIG\_PATH must be set.

Open the file /etc/profile (C:\msys\1.0\etc\profile) with your favorite editor (make sure it uses **LF** line endings) and add:

export PKG\_CONFIG\_PATH="/mingw/lib/pkgconfig"

export CVS\_RSH=ssh

export CFLAGS="… -march=i686"

export HOME LOGNAME MSYSTEM HISTFILE

./configure --prefix=/mingw

make

make install

* Installing to "/usr/local" should be avoided, "/mingw" should be used instead to avoid conflicts with msysdvlpr.

<https://github.com/niXman/mingw-builds/tree/master/scripts>

<https://github.com/niXman/mingw-builds>