**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. Ich verwende mingw32 (nicht das von Qt, sondern ein separates) und baue alles damit, achte gleichzeitig aber darauf, dass das Bauen mit mingw64 auch funktioniert, also dass bspw. überall die richtigen Verzeichnisse verwendet werden.

Laden Sie sich die aktuelle Version mit win32-Threads-Implementierung herunter. Prinzipiell sollten auch posix-Threads gleichwertig funktionieren, aber wir arbeiten schließlich unter Windows. Nach dem Entpacken des 1. Archivs muss das Verzeichnis mingw noch umbenannt werden, bspw. in mingw32 bzw. mingw64, damit das Entpacken des 2. Archivs mit dem gleichen Ziel-Verzeichnis nichts durcheinander bringt.

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

<http://sourceforge.net/projects/mingwbuilds/files/host-windows/releases/4.7.2/>

x32-4.7.2-release-win32-sjlj-rev10.7z Verzeichnis mingw umbenennen in mingw32

x64-4.7.2-release-win32-sjlj-rev10.7z Verzeichnis mingw umbenennen in mingw64

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: 5.0.2  
Datum: 12.03.2013  
<http://origin.releases.qt-project.org/digia/5.0.2/latest_successful/>

qt-windows-opensource-5.0.2-mingw47\_32-x86-offline-2013-03-12-454.exe

Die einfachste Variante ist es, das offizielle Komplettpaket von Qt zu installieren. 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 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/>

Unfortunately the start menu entries are only available for the account which installed Qt. Copy the "Qt 5.0.1" directory from the start menu 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:

Qt-5.0.1-x32.7z Verzeichnis Qt32-5.0.1

Qt-5.0.1-x64.7z Verzeichnis Qt64-5.0.1

Um die Binaries aus dem QtXY-Paketen benutzen zu können, muss 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:\Qt32-5.0.1\bin\qt.conf - Verzeichnis korrigieren!

C:\Qt64-5.0.1\bin\qt.conf - Verzeichnis korrigieren!

## 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-rev9.7z herunterladen und extrahieren. Das Verzeichnis msys irgendwo in einem Laufwerk ganz weit an der Wurzel installieren. Also beispielsweise als C:\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.

Die Datei /etc/fstab muss noch angepasst werden. Meine sieht 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!

## 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 desweiteren 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 Compilerflags 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 bereit stellt.

## Environment

Die Arbeitsumgebung benötigt einige Variablen, welche der Einfachheit halber per Funktion gesetzt werden. Dazu diesen Code per Cut+Paste einfügen oder einfach sourcen.

function oegenv {

if [ ! $# == 1 ]; then

echo "Usage: oegenv <32 or 64 bits as number>" # not $0

return # don’t use exit!

fi

local STDCFLAGS="-pipe -O2 -mms-bitfields -fomit-frame-pointer -I$BASEDIR/include"

if [ "$1" = "32" ]; then

export CFLAGS="-m32 -march=i686 -mtune=i686 $STDCFLAGS"

export LDFLAGS="-m32 -pipe -L$BASEDIR/lib"

export TARGETBITS="32"

export CONFIGURE\_HOST\_PARA="--host=i686-w64-mingw32"

elif [ "$1" = "64" ]; then

export CFLAGS="-m64 -march=nocona -mtune=core2 $STDCFLAGS"

export LDFLAGS="-m64 -pipe -L$BASEDIR/lib64"

export TARGETBITS="64"

export CONFIGURE\_HOST\_PARA="--host=x86\_64-w64-mingw32"

else

echo Wrong parameter: $1

return

fi

export BASEDIR="/open-egovernment"

export WORKDIR="/work"

export OEGDIR="/i/Projekte/berlios/open-egov"

export ARCHIVESDIR="${OEGDIR}/data/builder/archives"

export DOWNLOADSDIR="${OEGDIR}/data/builder/downloads"

export PATCHESDIR="${OEGDIR}/data/builder/patches"

export CXXFLAGS="$CFLAGS"

export CPPFLAGS=""

export LINGUAS="en de ar ru"

export LANG=de

export PATH\_TO\_7ZIP="/c/Programme/7-Zip/7z.exe"

}

function oegarchive {

if [ ! $# == 1 ]; then

echo "Usage: oegarchive \"name-version\""

return

fi

if [ "$1" = "" ]; then

echo "Error: name was empty"

return

fi

if [ "$TARGETBITS" = "" ]; then

echo "Error: unknown target platform (32 or 64 bits?), call oegenv()"

return

fi

local ARCHIVEFILENAME="$1-$TARGETBITS"

cd ${BASEDIR}

$PATH\_TO\_7ZIP a -mx9 -t7z -ms -mmt -w.. ../"$ARCHIVEFILENAME".7z .

$PATH\_TO\_7ZIP a -tzip -w.. ../${ARCHIVEFILENAME}.zip .

}

function oegcleanbase {

rm -rf ${BASEDIR}

mkdir ${BASEDIR}

}

## Vorbereitungen

oegenv 32

mkdir ${WORKDIR}

mkdir ${BASEDIR}

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}

tar xzf $DOWNLOADSDIR/libiconv-1.14.tar.gz

mv libiconv-1.14 libiconv-1.14-32

tar xzf $DOWNLOADSDIR/libiconv-1.14.tar.gz

mv libiconv-1.14 libiconv-1.14-64

oegenv 32

oegcleanbase

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

# ./configure --host=x86\_64-w64-mingw32

# 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 --host=x86\_64-w64-mingw32 --disable-shared

# make install-strip

# --with-libintl-prefix

# --with-libiconv-prefix

# --enable-relocatable

make

make install

oegarchive libiconv-1.14

chenv 64

oegcleanbase

cd ${WORKDIR}/libiconv-1.14-64

#configure --host=x86\_64 --build=i686 --prefix=${BASEDIR}

configure --host=x86\_64 --prefix=${BASEDIR} -–enable-shared

make

make install

oegarchive libiconv-1.14

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

make install  
make install DESTDIR=/open-egovernmentxxx

oegarchive libiconv-1.14

### 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$$

### xz

cd /work

/c/Programme/7-Zip/7z.exe x $DOWNLOADSDIR/xz-5.0.4.tar.xz

/c/Programme/7-Zip/7z.exe x xz-5.0.4.tar

#tar xf xz-5.0.4.tar

cd xz-5.0.4

configure --prefix=${BASEDIR} --disable-static --disable-xzdec --disable-lzmadec

make

make install-strip DESTDIR=/open-egovernment-xz

make clean

configure --prefix=${BASEDIR} --disable-shared --disable-nls --disable-encoders --enable-small --disable-threads

make -C src/liblzma

make -C src/xzdec

make -C src/liblzma DESTDIR=/open-egovernment-xz install-strip

cp -a extra /open-egovernment-xz/open-egovernment/share/doc/xz

oegarchive xz-5.0.4

### zlib

cd /work

tar xf $DOWNLOADSDIR/zlib-1.2.7.tar.gz

cd zlib-1.2.7

configure --help

mkdir -p /open-egovernment/bin

mkdir -p /open-egovernment/include

mkdir -p /open-egovernment/lib

make -fwin32/Makefile.gcc

make test testdll -fwin32/Makefile.gcc

BINARY\_PATH=/open-egovernment/bin INCLUDE\_PATH=/open-egovernment/include

LIBRARY\_PATH=/open-egovernment/lib make -fwin32/Makefile.gcc

BINARY\_PATH=/open-egovernment/bin INCLUDE\_PATH=/open-egovernment/include

LIBRARY\_PATH=/open-egovernment/lib make -fwin32/Makefile.gcc install

BINARY\_PATH=/open-egovernment/bin INCLUDE\_PATH=/open-egovernment/include

LIBRARY\_PATH=/open-egovernment/lib make -fwin32/Makefile.gcc install SHARED\_MODE=1

cd /open-egovernment/

oegarchive zlib-1.2.7

### termutils

cd /work/

tar xf /i/Projekte/berlios/open-egov/data/builder/downloads/termutils-2.0.tar.gz

cd termutils-2.0/

configure --help

#configure --prefix=/nonexistent -- DESTDIR wird ignoriert!

make

make install DESTDIR=/open-egovernment/

cd open-egovernment/

oegarchive termutils-2.0

### termcap

cd /work/

tar xf /i/Projekte/berlios/open-egov/data/builder/downloads/termcap-1.3.1.tar.gz

cd termcap-1.3.1/

configure --help

configure --prefix=/nonexistent --enable-install-termcap

make

make install DESTDIR=/open-egovernment/

configure --prefix=${BASEDIR}

make

make install

cd /open-egovernment/

oegarchive termcap-1.3.1

optional?

# --enable-install-termcap klappt nicht, per Hand machen

copy termcap.src to /etc/termcap

## Benötigte OSS-Pakete

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

### gettext

X

### iconv

X

NOTIZEN AB HIER

In früheren Versionen kam das Qt beigefügte MinGW mit verschiedenen Patches, wo tlw. unklar war, ob diese alle für Qt benötigt werden. Wie das jetzt ist, also ob, bzw. wenn ja, dann nur kleinere Bugfixes enthalten sind, habe ich noch nicht untersucht.

Liste der installierten Pakete (+Patches, Includes): C:\Qt\2010.02.1\mingw\manifest.txt

This directory contains the following packages and *modifications*:

binutils-2.19.1-mingw32-bin

mingwrt-3.15.2-mingw32-dev

mingwrt-3.15.2-mingw32-dll

w32api-3.13-mingw32-dev

gcc-core-4.4.0-mingw32-bin

gcc-core-4.4.0-mingw32-dll

gcc-c++-4.4.0-mingw32-bin

gcc-c++-4.4.0-mingw32-dll

gmp-4.2.4-mingw32-dll

libiconv-1.13-mingw32-dll-2

mingw32-make-3.81-20080326-3

mpfr-2.4.1-mingw32-dll

pthreads-w32-2.8.0-mingw32-dll

gdb and libexpat-1.dll

Modified with the patches in the directory \_patches.

Die Paketliste gibt erst einmal die Mindeststandards vor, die mit einer eigenen Entwicklungs­umgebung erreicht werden müssen. Die Patches (\*.diff) lassen sich manuell auf Wichtigkeit untersuchen. Unklar ist, inwieweit die beiliegenden Include-Dateien notwendig sind.

## MinGW

Version: 5.1.6

Verzeichnis „MinGW“: enthält nur die per Installer installierbaren Pakete.

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

runtime=mingwrt-3.17-mingw32-dev.tar.gz

runtimeDLL=mingwrt-3.17-mingw32-dll.tar.gz

w32api=w32api-3.14-mingw32-dev.tar.gz

binutils=binutils-2.20-1-mingw32-bin.tar.gz

core=gcc-core-3.4.5-20060117-3.tar.gz

gpp=gcc-g++-3.4.5-20060117-3.tar.gz

g77=gcc-g77-3.4.5-20060117-3.tar.gz

ada=gcc-ada-3.4.5-20060117-3.tar.gz

java=gcc-java-3.4.5-20060117-3.tar.gz

objc=gcc-objc-3.4.5-20060117-3.tar.gz

make=make-3.81-20090914-mingw32-bin.tar.gz

## MinGW-Updates (gcc)

Nun alle Pakete drüber installieren, welche neuer angeboten werden:

* mingwrt-3.18

Version: gcc-4.5.0-1

Datum: 2010-04-19

Verzeichnis: GCC-4.5.0-1

PREREQUISITES

- binutils 2.13 - installiert: 2.20-1

- mingw-runtime 3.13 - installiert: 3.17

- w32api 2.2 - installiert: 3.14

To get a complete GCC distribution, download the 'full' file. It includes the DLL prerequisites GMP, libiconv, MPFR, and support for all languages.

gcc-full-4.4.0-mingw32-bin.tar.lzma

If you do not want to download the entire compiler, you must download the required gmp, mpfr, and pthreads prerequisites, the 'core' bin and dll packages, plus the 'bin' and 'dll' packages of any other languages that you need.

GMP Runtime [REQUIRED]: gmp-4.2.4-mingw32-dll.tar.gz

libiconv Runtime [REQUIRED]: libiconv-1.13-mingw32-dll-2.tar.gz

MPFR Runtime [REQUIRED]: mpfr-2.4.1-mingw32-dll.tar.gz

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

Core (C) [REQUIRED]

gcc-core-4.4.0-mingw32-bin.tar.gz

gcc-core-4.4.0-mingw32-dll.tar.gz

Ada [OPTIONAL]

gcc-ada-4.4.0-mingw32-bin.tar.gz

gcc-ada-4.4.0-mingw32-dll.tar.gz

C++ [OPTIONAL]

gcc-c++-4.4.0-mingw32-bin.tar.gz

gcc-c++-4.4.0-mingw32-dll.tar.gz

Fortran [OPTIONAL]

gcc-fortran-4.4.0-mingw32-bin.tar.gz

gcc-fortran-4.4.0-mingw32-dll.tar.gz

Java [OPTIONAL]

gcc-java-4.4.0-mingw32-bin.tar.gz

gcc-java-4.4.0-mingw32-dll.tar.gz

Objective-C [OPTIONAL]

gcc-objc-4.4.0-mingw32-bin.tar.gz

gcc-objc-4.4.0-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/>.

Achtung: Das dem Qt-SDK beigelegte MinGW enthielt noch weitere DLLs (iconv, …). Diese sind hier noch nicht installiert worden. Da das MinGW-gettext sehr veraltet

## MSYS

Version: 1.0.11

Komplett und mit Vorgabe-Einstellungen per Installer installiert. 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

Version: 1.0.1 (01/2003) – verdammt alt!

Verzeichnis „MSYS”.

Im MinGW-Projekt unter „MSYS Supplementary Tools“. Version ist 1.0.1 (01/2003).

Install into C:\msys\1.0.

Provides perl and the autotools. perl is needed by texi2html to generate the documentation. autotools are needed by some extra libraries. The latest version of MSys DTK is badly outdated, I re-build my own autotools packages.

What does msysDTK contain?

The tools I’ve provided with msysDTK the following list of tools: 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. NOTE: I’ve only included the client tools of any package providing both server and client. It is not my goal for MSYS to provide server services. The cvs, openssl, openssh and inetutils are provided for remote operations with SourceForge. Perl is provided because autoconf, automake and libtool require it. Guile is provided because autogen requires it. And autogen is provided, just because it’s a nice tool.

## MySQL

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: a\*, kein remote Zugriff vom Netzwerk  
C:\Program Files\MySQL\MySQL Server 5.1\my.ini

Die Qt-Pfade (vgl. die Batch-Datei zum Setzen der Qt-Environment-Variablen qtvars.bat) in die ~/.profile (nicht .bashrc) o.ä. eintragen und ausprobieren:

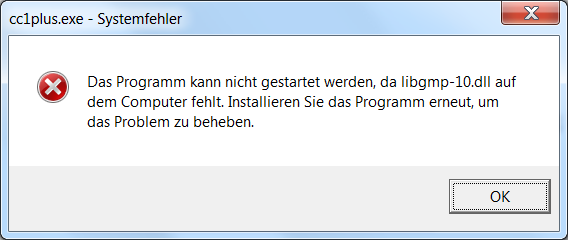
cd /qt/demos

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

../path/to/configure --prefix=/mingw && make && make install

* 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 now part of MSYS 1.0.11, installed as "make"

As all the libraries will be installed in /mingw, PKG\_CONFIG\_PATH must be set. Optionally, if you want to get the cvs repository as a developer, CVS\_RSH must also 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:

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

CVS\_RSH=ssh

You might also want to set CFLAGS. Safe options for MinGW are:

CFLAGS="… -march=i686"

export HOME LOGNAME MSYSTEM HISTFILE

export CFLAGS PKG\_CONFIG\_PATH CVS\_RSH

* To install 3rd party library and applications which uses the autotools build system the following commands are often used.

./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>