EPSol Compilation

# CMAKE Makefile

## Win32 Environment

### 1.1.1 From CMD

set CXX=x86\_64-w64-mingw32-g++.exe

set CC=x86\_64-w64-mingw32-gcc.exe

set PATH=D:\Luciano\Programas\x86\_64-4.9.1-posix-seh-rt\_v3-rev0\mingw64\bin;D:\Luciano\Programas\cmake-2.8.12.2-win32-x86\bin;%PATH%

Options

1. cmake ..\..\Src
2. PATH COMPLETO..cmake –D CMAKE\_CXX\_COMPILER=%CXX% -G “CodeBlocks – MinGW Makefiles” **PATH**

SI NO SE COLOCA PATH COMPLETO NO ANDA BIEN LA OPCION -G

### 1.1.2 From CYGWIN Environment

# PETSC Compilation

## Windows build from Cygwin terminal

64 bits MinGW compiler

export CXX=x86\_64-w64-mingw32-g++.exe

export CC=x86\_64-w64-mingw32-gcc.exe

cmake SRC\_PATH

cmake /cygdrive/c/Users/samsung/Dropbox/Numerico/MecSol/EPSol/Src

# PETSc Compilation

To compile with external mingw 32 bits compilers

Set environmental paths

To compile with stardard cygwin compilers

export PATH=/usr/bin

To compile with mingw external compilers

Example: if Mingw is in c:/MinGW

export PATH=/cygdrive/c/MinGW/bin:/usr/bin

export PATH=/cygdrive/c/MinGW/bin/:/usr/i686-w64-mingw32/bin:/usr/i686-w64-mingw32/sys-root/mingw/bin:/usr/local/bin:/usr/bin:/bin:

Simple without fortran

Sequential, without: fc, threads, mpi, shared

./configure --with-gcc=x86\_64-w64-mingw32-gcc.exe --with-cxx=x86\_64-w64-mingw32-g++.exe --with-cpp=x86\_64-w64-mingw32-cpp.exe --with-fc=0 --with-debugging=0 --with-mpi=0 --download-f-blas-lapack=1 --useThreads=0 --with-shared-libraries=0

With i686 compilers

./configure --with-gcc=i686-w64-mingw32-gcc.exe --with-cxx=i686-w64-mingw32-g++.exe --with-cpp=i686-w64-mingw32-cpp.exe --with-fc=0 --with-debugging=0 --with-mpi=0 --download-f-blas-lapack=1 --useThreads=0 --with-shared-libraries=0 --with-x=0

With external Mingw compilers

If --with-xxx compiler is specified, the following message appears

Could not locate a functional C compiler

Definitely....

1) Add --with-ar=/usr/bin: The mingw archiver is in conflict with cygwin archiver (from devel package)

2) To can set --with-cc instead of -CC: set bash

export PETC\_DIR=$PWD

export PATH=/cygdrive/c/MinGW/bin/:/usr/i686-w64-mingw32/bin:/usr/i686-w64-mingw32/sys-root/mingw/bin:/usr/local/bin:/usr/bin:/bin:/cygdrive/c/MinGW/mingw32/bin:$PATH

With cygwin mingw compilers:

./configure -CC=x86\_64-w64-mingw32-gcc.exe -CXX=x86\_64-w64-mingw32-g++.exe -CPP=x86\_64-w64-mingw32-cpp.exe --with-fc=0 --with-debugging=0 --with-mpi=0 --download-f-blas-lapack=1 --useThreads=0 --with-shared-libraries=0 --with-x=0 --with-clanguage=cxx --with-ar=/usr/bin

With mingw compilers

With -CC and -CXX

./configure -CC=mingw32-gcc.exe -CXX=mingw32-g++.exe -CPP=x86\_64-w64-mingw32-cpp.exe --with-fc=0 --with-debugging=0 --with-mpi=0 --download-f2cblaslapack=1 --useThreads=0 --with-shared-libraries=0 --with-x=0 --with-clanguage=cxx --with-ar=/usr/bin/ar

./configure --with-cc=mingw32-gcc.exe --with-cxx=mingw32-g++.exe -CPP=x86\_64-w64-mingw32-cpp.exe --with-fc=0 --with-debugging=0 --with-mpi=0 --download-f2cblaslapack=1 --useThreads=0 --with-shared-libraries=0 --with-x=0 --with-clanguage=cxx --with-ar=/usr/bin/ar

FINAL

simply with cygwin compilers

1) export PETC\_DIR=$PWD

2) export PATH=/usr/local/bin:/usr/bin:/bin:$PATH

3) ./configure --with-cc=gcc --with-cxx=g++ -CPP=cpp --with-fc=0 --with-debugging=1 --with-mpi=0 --download-f2cblaslapack=1 --useThreads=0 --with-shared-libraries=0 --with-x=0 --with-clanguage=cxx --with-ar=/usr/bin/ar

-----------

(DESCARGAR PACKAGE BINUTILS de CYGWIN PARA EL AR)

With cygwin mingw 64 bits compilers THIS WORKS

0) cd PEtSC DIRECTory

1) export PETSC\_DIR=$PWD

2) export PETSC\_ARCH=arch-mswin-c-debug-debug-sinmpi-64

3) export PATH=/cygdrive/c/MinGW/bin/:/usr/i686-w64-mingw32/bin:/usr/i686-w64-mingw32/sys-root/mingw/bin:/usr/local/bin:/usr/bin:/bin:/cygdrive/c/MinGW/mingw32/bin:$PATH

4) ./configure --with-cc=x86\_64-w64-mingw32-gcc.exe --with-cxx=x86\_64-w64-mingw32-g++.exe --with-fc=0 --with-mpi=0 --download-f2cblaslapack=1 --useThreads=0 --with-ar=/usr/bin/ar

------------------------------------------------------------------

* CASO 32 bits

export PATH=/cygdrive/c/MinGW/bin/:/usr/i686-pc-mingw32/bin:/usr/i686-pc-mingw32/sys-root/mingw/bin:/usr/local/bin:/usr/bin:/bin:$PATH

./configure --with-cc=i686-pc-mingw32-gcc.exe --with-cxx=i686-pc-mingw32-g++.exe --with-fc=0 --with-mpi=0 --download-f2cblaslapack=1 --useThreads=0 --with-ar=/usr/bin/ar --with-debugging=0

----------------------------------------------------------------

--with-shared-libraries=1

DEBUG MODE: --with-debugging=0

Shared libraries NO FUNCIONA, ASI QUE LA dll es epsol, este siempre linkea estatico

Cross Compiling: --host=x86\_64-w64-mingw32 (previously export PATH=/home/lucho/mxe/usr/bin:$PATH)

If you have blaslapack

./configure --with-cc=x86\_64-w64-mingw32-gcc.exe --with-cxx=x86\_64-w64-mingw32-g++.exe --with-fc=0 --with-mpi=0 --with-blas-lapack-dir=/petsc-3.4.4/externalpackages/f2cblaslapack-3.4.1.q --useThreads=0 --with-ar=/usr/bin/ar --with-shared-libraries=1

5) Like log indicates: make PETSC\_DIR=/petsc-3.4.4 PETSC\_ARCH=arch-mswin-c-debug-debug-x|

-OJO!

Colocando el export PETSC\_DIR

Luego Sale

=========================================

Now to check if the libraries are working do:

make PETSC\_DIR=/petsc-3.4.4 PETSC\_ARCH=arch-mswin-c-debug-debug-sinmpi-64 test

=========================================

El que anda es (pero hay que poner el export PETSC\_DIR=$PWD):

--with-cc=x86\_64-w64-mingw32-gcc.exe --with-cxx=x86\_64-w64-mingw32-g++.exe --with-fc=0 --with-mpi=0 --download-f2cblaslapack=1 --with-ar=/bin/ar --useThreads=0

With cygwin mingw 32 bits compilers THIS WORKS

0) cd PEtSC DIRECTory

1) export PETSC\_DIR=$PWD

2) export PETSC\_ARCH=arch-mswin-c-debug-debug-sinmpi-64

3) export PATH=/cygdrive/c/MinGW/bin/:/usr/local/bin:/usr/bin:/bin:/cygdrive/c/MinGW/mingw32/bin:/cygdrive/c/MinGW/libexec/gcc/mingw32:$PATH

4) ./configure --with-cc=mingw32-gcc.exe --with-cxx=mingw32-g++.exe --with-fc=0 --with-mpi=0 --download-f2cblaslapack=1 --useThreads=0 --with-ar=/usr/bin/ar

5) LIKE LOG INDICATES: make PETSC\_DIR=/petsc-3.4.4 PETSC\_ARCH=arch-mswin-c-debug-debug-sinmpi-64

If you have lapack

./configure --with-cc=mingw32-gcc.exe --with-cxx=mingw32-g++.exe --with-fc=0 --with-debugging=1 --with-mpi=0 --with-blas-lapack-dir=/petsc-3.4.4/externalpackages/f2cblaslapack-3.4.1.q --useThreads=0 --with-shared-libraries=0 --with-x=0 --with-clanguage=cxx --with-ar=/usr/bin/ar

5) Like log indicates: make PETSC\_DIR=/petsc-3.4.4 PETSC\_ARCH=arch-mswin-c-debug-luciano all

another option is:

export CC= Whatever

./configure CC=$CC

-------------------- CAMBIOS LUEGO DE CONFIGURAR (ANTES DE MAKE) ------------------------------

ESTO EN VERDE NO DEBE HACERSE, PORQUE SE MUESTRA ERROR SI SE QUIERE LINKEAR COMO SHARED

SE saco la carpeta win32 de

src/sys/classes/draw/impls/

Y modifique el makefile de este directorio sacandole la inclusion

APARTE, ANTES QUE NADA (UNA VEZ BAJADA LA LIBRERIA), se hacen estos dos cambios:

1. TB MODIFIQUE /petscdir/src/dm/impls/plex.c agregandole el include petscsys.h
2. Por ultimo a /petscdir/petscsys.h le DESCOMENTÉ un inline en línea 2339:

PETSC\_STATIC\_INLINE PetscErrorCode PetscSegBufferGetInts

------------------------ LUEGO DE COMPILAR (SI NO SE QUIERE WINDOWS) ---------------

lueggo se cambio dps de configurar el archivo petscconf.h (dentro de debug)

se saco el def de

//MODIFIED BY LUCIANO, SET TO 0

//#ifndef PETSC\_USE\_WINDOWS\_GRAPHICS

//#define PETSC\_USE\_WINDOWS\_GRAPHICS 0

//#endif

Si no se hace esto no se encuentra PetscDrawCreate\_Win32 no se encuentra

## PETSC with cross compiling environment (MXE)

Si tengo la librería MPI: --with-mpi-lib=PATH A LA LIBRERIA

--with-mpi-lib

# WxWidgets

export PATH=/home/lucho/mxe/usr/bin:$PATH

../configure --host=x86\_64-w64-mingw32 --enable-static --enable-monolithic --enable-unicode

# General Cross Compiling

## 2.1

## Tutorial

### Step 1: Requirements and Download

First, you should ensure that your system meets MXE's [requirements](http://mxe.cc/#requirements). You will almost certainly have to install some stuff.

When everything is fine, download the [current stable version](http://mxe.cc/#download):

git clone -b stable https://github.com/mxe/mxe.git

If you don't mind installing it in your home directory, just skip the following step and go straight to step 3.

MXE builds and installs everything under the same top-level directory and is not relocatable after the first packages are built.

### Step 2: System-wide Installation (optional)

Now you should save any previous installation of the MXE. Assuming you've installed it under /opt/mxe (any other directory will do as well), you should execute the following commands:

su

mv /opt/mxe /opt/mxe.old

exit

Then you need to transfer the entire directory to its definitive location. We will assume again you use /opt/mxe, but feel free to use any other directory if you like.

su

mv mxe /opt/mxe

exit

We're almost done. Just change to your newly created directory and get going:

cd /opt/mxe

### Step 3: Build MXE

Enter the directory where you've downloaded MXE. Now it depends on what you actually want – or need.

If you choose to enter:

make

you're in for a long wait, because it compiles [a lot of packages](http://mxe.cc/#packages). On the other hand it doesn't require any intervention, so you're free to do whatever you like – like watch a movie or go for a night on the town. When it's done you'll find that you've installed a very capable Win32 cross compiler onto your system.

If you only need the most basic tools you can also use:

make gcc

and add any additional packages you need later on. You can also supply a host of packages on the [command line](http://mxe.cc/#usage), e.g.:

make gtk lua libidn

Targets can also be specified on the command line. By default, only i686-pc-mingw32 is built, but you can build your toolchain(s) of choice with:

make MXE\_TARGETS='x86\_64-w64-mingw32 i686-w64-mingw32'

or by adjusting the MXE\_TARGETS variable in settings.mk.

You'll always end up with a consistent cross compiling environment.

If you have trouble here, please feel free to contact the mxe team through the [issue tracker](https://github.com/mxe/mxe/issues) or [mailing list](https://lists.nongnu.org/mailman/listinfo/mingw-cross-env-list).

After you're done it just needs a little post-installation.

### Step 4: Environment Variables

Edit your .bashrc script in order to change $PATH:

export PATH=/where MXE is installed/usr/bin:$PATH

You may be tempted to also add $(TARGET)/bin to your path. You**never** want to do this, the executables and scripts in there will cause conflicts with your native toolchain.

In case you are using custom $PKG\_CONFIG\_PATH entries, you can add separate entries for cross builds:

export PKG\_CONFIG\_PATH="entries for native builds"

export PKG\_CONFIG\_PATH\_i686\_pc\_mingw32="entries for MXE builds"

Remember to use i686-pc-mingw32-pkg-config instead of pkg-config for cross builds. The Autotools do that automatically for you.

Note that any other compiler related environment variables (like $CC, $LDFLAGS, etc.) may spoil your compiling pleasure, so be sure to delete or disable those.

For the most isolated and repeatable environment, use a white-list approach:

unset `env | \

grep -vi '^EDITOR=\|^HOME=\|^LANG=\|MXE\|^PATH=' | \

grep -vi 'PKG\_CONFIG\|PROXY\|^PS1=\|^TERM=' | \

cut -d '=' -f1 | tr '\n' ' '`

Congratulations! You're ready to cross compile anything you like.

### Step 5a: Cross compile your Project (Autotools)

If you use the [Autotools](https://www.lrde.epita.fr/~adl/autotools.html), all you have to do is:

./configure --host=i686-pc-mingw32

make

If you build a library, you might also want to enforce a static build:

./configure --host=i686-pc-mingw32 --enable-static --disable-shared

make

Don't worry about a warning like this:

configure: WARNING: If you wanted to set the --build type, don't use --host.

If a cross compiler is detected then cross compile mode will be used.

Everything will be just fine.

### Step 5b: Cross compile your Project (CMake)

If you have a [CMake](http://www.cmake.org/) project, you can use the provided toolchain file:

cmake ... -DCMAKE\_TOOLCHAIN\_FILE=/where MXE is installed/usr/i686-pc-mingw32/share/cmake/mxe-conf.cmake

### Step 5c: Cross compile your Project (Qt)

If you have a [Qt](https://qt-project.org/) application, all you have to do is:

/where MXE is installed/usr/i686-pc-mingw32/qt/bin/qmake

make

Note that Qt 4 is in the "qt" subdirectory. Qt 5 is in the "qt5" subdirectory and its qmake can be invoked similarly.

If you are using Qt plugins such as the svg or ico image handlers, you should also have a look at the [Qt documentation about static plugins](https://qt-project.org/doc/qt-4.8/plugins-howto.html" \l "static-plugins).

Note the sql drivers (-qt-sql-\*) and the image handlers for jpeg, tiff, gif and mng are built-in, not plugins.

### Step 5d: Cross compile your Project (Makefile)

If you have a handwritten Makefile, you probably will have to make a few adjustments to it:

CC=$(CROSS)gcc

LD=$(CROSS)ld

AR=$(CROSS)ar

PKG\_CONFIG=$(CROSS)pkg-config

You may have to add a few others, depending on your project.

Then, all you have to do is:

make CROSS=i686-pc-mingw32-

That's it!

### Step 5e: Cross compile your Project (OSG)

Using static OpenSceneGraph libraries requires a few changes to your source. The graphics subsystem and all plugins required by your application must be referenced explicitly. Use a code block like the following:

#ifdef OSG\_LIBRARY\_STATIC

USE\_GRAPHICSWINDOW()

USE\_OSGPLUGIN(<plugin1>)

USE\_OSGPLUGIN(<plugin2>)

...

#endif

Look atexamples/osgstaticviewer/osgstaticviewer.cpp in the OpenSceneGraph source distribution for an example. This example can be compiled with the following command:

i686-pc-mingw32-g++ \

-o osgstaticviewer.exe examples/osgstaticviewer/osgstaticviewer.cpp \

`i686-pc-mingw32-pkg-config --cflags openscenegraph-osgViewer openscenegraph-osgPlugins` \

`i686-pc-mingw32-pkg-config --libs openscenegraph-osgViewer openscenegraph-osgPlugins`

The i686-pc-mingw32-pkg-config command from MXE will automatically add -DOSG\_LIBRARY\_STATIC to your compiler flags.

### Further Steps

If you need further assistance, feel free to join the [mailing list](https://lists.nongnu.org/mailman/listinfo/mingw-cross-env-list) where you'll get in touch with the MXE developers and other users.

Hi all,

after a long time I've finally succeeded in compiling a 64 bit parallel version of Petsc for Windows.

All tries with MPICH2 failed. It is known to have problems with newer 64 bit Windows (e.g. with UAC).

Therefore I gave Microsoft's MPI (which is also free of charge) a try.

Unfortunately there are some problems when using mingw compilers. (Some Fortran symbol

names are not compatible). So it does not work out of the box.

But the problems can easily be solved.

Here is the way it works:

Preparing Microsoft's MPI to work with x86\_64-w64-mingw32-gfortran

0.) You need Cygwin. Please make sure that the Devel tools and Python are installed.

1.) Download Microsoft's "HPC Pack 2008 R2 MS-MPI Redistributable Package":

<http://www.microsoft.com/en-us/download/details.aspx?id=14737>

2.) Install it in a Directory without any spaces in the path/folder name

3.) Insert the following line at the beginning of the file InstallationPath\Inc\mpi.h

#include <stdint.h>

4.) In the file InstallationPath\Inc\mpif.h

replace INT\_PTR\_KIND()

by 8

5.) Copy C:\Windows\System32\msmpi.dll to InstallationPath\Lib\amd64\msmpi.dll

6.) In a Cygwin terminal please enter the following:

cd /cygdrive/c/your/HPC/installation/path/Lib/amd64

gendef msmpi.dll

x86\_64-w64-mingw32-dlltool -d msmpi.def -l libmsmpi.a -D msmpi.dll

This creates a library (libmsmpi.a) which now works with x86\_64-w64-mingw32-gfortran.

Compiling Petsc

7.) Download Petsc v3.1-p8: petsc-3.1-p8.tar.gz<<http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.1-p8.tar.gz>> (or <http://www.mcs.anl.gov/petsc/download/index.html>)

8.) Untar it (In a Cygwin terminal go to the folder where you've downloaded petsc-3.1-p8.tar.gz<<http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.1-p8.tar.gz>>

and type: tar -xzvf petsc-3.1-p8.tar.gz) and enter this directory (cd petsc-3.1-p8).

9.) Edit the file src/sys/memory/mal.c:

Change in line 39 (unsigned long) to (unsigned long long)

10.) Edit the file include/mpiuni/mpi.h:

Change in line 112

#define MPIUNI\_INTPTR long

to:

#define MPIUNI\_INTPTR long long

11.) Set the PATH variable in the Cygwin terminal:

export PATH=/usr/x86\_64-w64-mingw32/bin:/usr/x86\_64-w64-mingw32/sys-root/mingw/bin:/usr/local/bin:/usr/bin:/bin: / cygdrive/c/your/HPC/installation/path/Bin:$PATH

12.) Configure PETSc with the command given below in the Cygwin shell. Please adapt your mpi path and choose with-scalar-type=complex

or with-scalar-type=real according your needs.

Please note: GetDP can solve problems with real or complex DOFs regardless the scalar-type of Petsc. In this context the scalar-type is just an optimization in terms of speed and memory consumption.

./configure --CC=x86\_64-w64-mingw32-gcc.exe --CXX=x86\_64-w64-mingw32-g++.exe --FC="x86\_64-w64-mingw32-gfortran.exe -fno-range-check" --CPP=x86\_64-w64-mingw32-cpp.exe --with-debugging=0 --with-clanguage=cxx --with-shared=0 --with-x=0 --useThreads=0 --download-f-blas-lapac=ifneeded --download-mumps=ifneeded --download-parmetis=ifneeded --download-scalapack=ifneeded --download-blacs=ifneeded --with-scalar-type=complex --with-mpi-include=/cygdrive/c/your/HPC/installation/path/Inc --with-mpi-lib=/cygdrive/c/your/HPC/installation/path /Lib/amd64/libmsmpi.a

13.) Before Petsc is compiled the file ./cygwin-cxx-opt/include/petscconf.h has to be edited.

Replace ( BEWARE: check the number of underscores, don't touch to PETSC\_HAVE\_\_SLEEP )

#ifndef PETSC\_HAVE\_SLEEP

#define PETSC\_HAVE\_SLEEP 1

#endif

with

#undef PETSC\_HAVE\_SLEEP

and replace

#ifndef PETSC\_HAVE\_GETPAGESIZE

#define PETSC\_HAVE\_GETPAGESIZE 1

#endif

with

#undef PETSC\_HAVE\_GETPAGESIZE

14.) Now let's compile it.

make PETSC\_DIR=/your/path/to/petsc-3.1-p8 PETSC\_ARCH=cygwin-cxx-opt all

15.) Drink some coffee and pray that everything works ;-)

That's all.

Hopefully this is helpful for some of you.

Have a nice day!

Michael

Michael Asam

Infineon Technologies AG

ATV BP PD1 M1, Munich

Infineon Technologies AG

Vorsitzender des Aufsichtsrats: Wolfgang Mayrhuber

Vorstand: Peter Bauer (Vorsitzender), Dominik Asam, Arunjai Mittal, Dr. Reinhard Ploss

Sitz der Gesellschaft: Neubiberg

Registergericht: München HRB 126492