

Compiling & Building MEX files for Matlab on a MacBook Pro using Xcode 2.4.1, Intel Compiler, Intel Performance Primitives (IPP) and Intel Math Kernel Libraries (MKL)

Author: Marco Zuliani

Version: 1.1.0

Last Modified: 7 December 2007 [added section **The Right Mac O-type**]

Introduction

In these notes I will describe my first (successful) attempt to compile and link MEX files for Matlab using Xcode IDE on a Intel Core 2 MacBook Pro (but hopefully the procedure should be general enough to adapt to other scenarios). Since I am new to the Mac's world (after many years using solely Windows and a little bit of Linux) it is likely that this notes are not totally accurate or that some issues can be solved much easily. Do not hesitate to contact me and to send suggestions/corrections.

The Right Mac O-type

During experimentations with MEX files, one may need to recompile the MEX function (clearly without restarting the Matlab session). Under Windows it is enough to issue a command `clear functions` to allow the linker to rebuild the MEX file (which is actually a `.dll`). I thought that a similar approach would have worked for the Mac as well. In the first version of this document I suggested to set the **Mac-O Type** to `Dynamic Library`. It did not take a long time before I realized that this was not a good choice... The original MEX file was somehow *persistent* in memory even after recompiling the library and issuing a `clear functions` command from Matlab: to get the newly compiled version to work I had to restart Matlab each and every time. After browsing a little bit documentation files, blogs, and whatever else, I learned that there exists different flavors of dynamic libraries and one needs to be careful which one to choose. Issuing a command `file Whatever.mexmaci` it turned out that the *persistent* MEX file build using my approach was a dynamically linked shared library. On the other hand, checking the result of the compilation via a `mex Whatever.c` script in Matlab, I found that the MEX file was actually a `bundle`. At this point the solution seemed to be quite easy... just change the **Mac-O Type** field from `Dynamic Library` to `Bundle`. Unfortunately things did not work out that smoothly. At this point Xcode started giving me a hard time during the linking process, failing with this obscure error message `-compatibility_version only allowed with -dynamiclib`. Luckily I managed to find a post (see references) from the Xcode archive that helped me in setting up a

workaround for this problem. This entire procedure has now been incorporated in the steps described in the next section. **Moral of the story:** feel free to recompile your MEX file without restarting Matlab: the newly compiled version will be correctly loaded!

The Basic Steps to Compile and Build a MEX Function Dynamically Linked to IPP and/or MKL

Suppose the MEX source file is `$(MEX_Project)\MEX_Project\MEX_Project.cpp`, and that the installation paths for Matlab and for the Intel libraries are:

- Matlab (usually `/opt/matlab`)
- IPP (usually `/Library/Frameworks/Intel_IPP.framework/`)
- MKL (usually `/Library/Frameworks/Intel_MKL.framework/`)

The fundamental steps to compile and build your MEX file (in debug mode, since this is the default when you create a new project) are the following:

1. Create a new project from Xcode: **File->New Project**. Select **Dynamic Library->BSD Dynamic Library**. Select the path consistently with `MEX_Project.cpp`
2. In the **Groups & Files** pane add to **Source** all the `.c`, `.cpp`, `.h`, `.hpp` files that compose your project
3. Select the target file in the **Groups & Files** pane in the project window (**Target** item) and press the button **info** (or right click and **Get Info**)
4. Select the **Build** tag in the Target Info window.
5. Make sure the **Configuration** field is set to `All Configurations` and the **Collection** is set to `All Collections` or `All Settings`
6. In the **Header Search Paths** field add: `$(Matlab)/extern/include $(IPP)/Headers $(MKL)/Headers` which, in case of standard paths, should read as: `/opt/matlab/extern/include /Library/Frameworks/Intel_IPP.framework/Headers /Library/Frameworks/Intel_MKL.framework/Headers` (this is probably what you want to copy & paste...)
7. In the **Library Search Path** field add: `$(Matlab)/bin/maci $(IPP)/Libraries $(MKL)/Libraries/32` which, in case of standard paths, should read as: `/opt/matlab/bin/maci /Library/Frameworks/Intel_IPP.framework/Libraries /Library/Frameworks/Intel_MKL.framework/Libraries/32` (this is probably what you want to copy & paste...)
8. Make sure that the **Mac-O Type** field is set to `Bundle`
9. Set the **Current Library Version** field to empty (i.e. the field should be blank, no options)
10. Set the **Compatibility Version** field to empty
11. In the **Other Linker Flags** add the standard Matlab libraries that you need, e.g.:

- lmex -lmx (you may also need -lmat)
- 12. Read in the IPP documentation which is the library file containing the function you want to utilize (usually it has the same name of the header file that contains the function declaration) and in the **Other Linker Flags** add: `-lipp(whatever) -lippcore -lguide`.
- 13. Similar considerations apply if you are using MKL. If you plan to use any of the MKL functions add in the **Other Linker Flags** the library: `-lmkl_intel`
- 14. Mark the **Perform Single-Object Prelink** checkbox
- 15. Make sure that the **Executable Prefix** field is empty (unless you want some prefix to you function such as MEX...)
- 16. Set the **Executable Extension** field to `mexmaci` (which is the standard extension for a Matlab MEX file compiled for a Intel Mac)
- 17. If you need to define some preprocessor symbols, add them to the **Preprocessor Macros** field (if you are using GNU C/C++).
- 18. In the Project window press the **Build** button to build the MEX function. If there are no errors the MEX function should compile and link just fine.
- 19. If you get some warnings like `can't open dynamic library: something-related-to-Matlab` there is a good chance that the MEX file will still be executed correctly. As of today, I am not sure why this happens...
- 20. If in the Project window the **Active Build Configuration** is set to `whatever` (by default should be set to `Debug`) you should find your MEX function in: `$(MEX_Project)\MEX_Project\build\whatever`

Optimizing with Intel Compiler

The following section deals with the configuration of the Intel Compiler inside the Xcode IDE. Before continuing make sure that the MEX function is built successfully in debug mode using the default compiler (probably GNU C/C++) following the steps listed in the previous section. Let's now enable the Intel Compiler to build the code in release mode:

1. Select the target file in the **Groups & Files** pane in the project window (**Target** item) and press the button **info** (or right click and **Get Info**)
2. Click **Rules** in the Target Info window.
3. Click the **+** button at the bottom, left-hand corner of the Target Info window
4. From the **Process** pull-down menu choose: `C source files`
5. From the **Using** pull-down menu choose: `Intel® C++ Compiler 10.1` (you might have a different version, but this should not matter...)
6. From the **Process** pull-down menu choose: `C++ source files`
7. From the **Using** pull-down menu choose: `Intel® C++ Compiler 10.1`
8. In the Project window from the **Active Build Configuration** select: `Release`
9. Click **Build** in the Target Info window.
10. Set the **Configuration** field to `Release` and the **Collection** field to

Architectures

11. Check the **Architectures** field to set the target that is suitable for you, e.g. `i386` (I believe that the `ppc` option will create troubles if you use Intel libraries...)
12. In the Project window press the **Build** button to build the MEX function. If there are no errors the MEX function should compile in release mode using the Intel compiler and link just fine.

Now we are going to set the optimization flags for the Intel compiler. The goal is to provide *general guidelines* to set the flags that potentially could generate fast functions (this does not necessarily mean that the options described in the following are the best ones or my any means exhaustive. For your specific application refer to the compiler manuals to get some deeper insights and always keep in mind D. Knuth's thought "Premature optimization is the root of all evil (or at least most of it) in programming."):

1. Select the target file in the **Groups & Files** pane in the project window (**Target** item) and press the button **info** (or right click and **Get Info**)
2. Click **Build** in the Target Info window.
3. Set the **Configuration** field to `Release` and the **Collection** field to `Intel® C++ Compiler 10.1` (you might have a different version, but this should not matter...)
4. Make sure the **Generate Debug Information** is `unchecked`
5. Set the **Optimization Level** to `Maximize Speed Plus High Level Optimizations (-O3)`
6. Make sure the **Enable Prefetch Insertion** is `checked`
7. Set **Use Intel(R) IA-32 Instruction Set Extension** to `Intel Core(TM)2 Duo`
8. Set **Floating Point Model** to `Fast=2 (-fp-model fast=2)`
9. Set **Floating Point Speculation** to `Fast Speculation (-fp-speculationfast)`
10. In the Project window press the **Build** button to build the optimized MEX function. If there are no errors the file should compile in optimized release mode using the Intel compiler and link just fine.

Setting the Environment Variables

In order to have Matlab successfully executing the function you just created, you need to set the environmental variable that points where the IPP and MKL dynamic libraries are located. From the X11 terminal issue the following command before launching Matlab:

```
- export DYLD_LIBRARY_PATH=$(IPP)Libraries:$(MKL)Libraries/32
```

which, in case of standard paths, should read as:

```
- export DYLD_LIBRARY_PATH=/Library/Frameworks/  
Intel_IPP.framework/Libraries:/Library/Frameworks/  
Intel_MKL.framework/Libraries/32
```

Then launch Matlab and you should be able to use your MEX function which is located in `$(MEX_Project)\MEX_Project\build\whatever`, where `whatever` is the name of the active configuration.

Some Other Useful References

In this section I will list some other useful references that provide information pertinent to the topics covered in these notes:

- General information regarding MEX files: <http://www.mathworks.com/support/tech-notes/1600/1605.html>
- Fixing a bug in Xcode: <http://www.cocoabuilder.com/archive/message/xcode/2006/1/15/2883>
- Using the Intel Compiler and Xcode: <http://www.intel.com/support/performancetools/c/mac/sb/CS-025886.htm>
- Debugging MEX files within the Xcode IDE on an Intel Mac: <http://www.mathworks.com/support/solutions/data/1-3U6RYL.html?solution=1-3U6RYL>