Using the MatLab Engine from Visual Studio's C++ Compiler

Document: 202

www.signal-processing.net

andreas_dsp@hotmail.com

Author: Andreas Schwarzinger

Date: June 14, 2019

Introduction

Being able to open and run MatLab while executing an application written in a different language can be quite helpful, especially in testing applications. In this document, we will step the reader through the process of launching and executing MatLab scripts from the Visual Studio C++ compiler. We will show this for both the 64-bit MatLab 2018b and the 32-bit MatLab2015b versions. After the 2015b release, MatLab no longer ships a 32-bit version of MatLab, but given that a great deal of software is still written for 32-bit applications, it is helpful to cover the use of the MatLab engine for the legacy 32-bit version. Accessing the MatLab engine for these two versions differs mainly in the way we set up the C++ project settings. In addition, some of the function calls in the MatLab Matrix API (mex.h or engine.h) exposed to the C++ environment have changed slightly.

Additional References

Discussion regarding the C++ functionality provided via the *mex.h* header file:

https://www.mathworks.com/help/matlab/cc-mx-matrix-library.html

Document 201: Creating and testing Mex Functions in Visual Studio

Setting Up the Visual Studio Project

If you have worked with mex function before (see document 201 of this series), then you may already be aware of the fact that setting up the your C++ project is half the battle. In this section, we step you through the process of properly configuring your project for both the x86 and x64 configurations. In this example, we assume that both MatLab2018b and MatLab 2015b are installed on your machine. It is quite typical to declare environment variables that abbreviate the root path at which MatLab is installed, and thus save yourself unnecessary typing and provide some degree of version independence. Let us look at the environment variables we will use for this exercise.

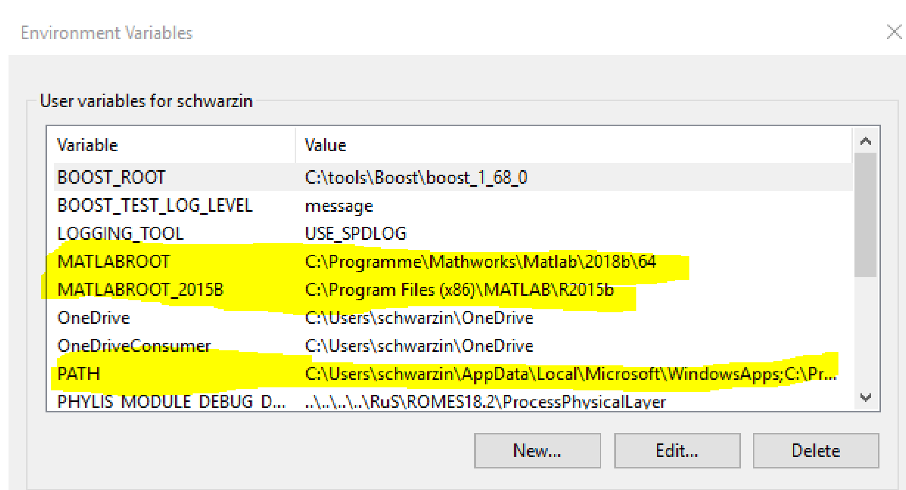

Figure 1: Environment Variable Settings

When configuring your project settings it is convenient to use the variable MATLABROOT to indicate the root path of your installation for portability reasons. For example, you may want to share the project with another developer who is using a different MatLab version and therefore has differences in his directory structure or nomenclature. Your colleague simply declares the same environment variable on his machine but changes its value to his MatLab root directory and everything should seamlessly compile for him. In our case, we define two separate root path names, one for MatLab2018b and the other for MatLab 2015b.

Note that we have highlighted the PATH variable as well. It is common for applications to launch other executables or DLLs during their operations without necessarily knowing where they are located on your computer. The operating system has a set of directory, which it will search in order to find the name of the executable or DLL that is being launched. When we load the MatLab engine, there are quite a few DLLs that need to be loaded (libmx.dll, libmat.dll, libeng.dll … etc) and providing their location is necessary for the overall process to work. The DLLs, provided by MatLab, are in the following locations.

2018b → C:\Programme\Mathworks\Matlab\2018b\64\bin\win64

2015b → C:\Program Files (x86)\MATLAB\R2015b\bin\win32

You will have to set one of these paths if you want to run your compiled C++ application in stand-alone mode. However, while you are developing in Visual Studio, you can also set the path as indicated below.

→ Temporarily setting the DLL search path from your project. To let your application know the proper location from which to load MatLab DLLs, we provide the following project setting. The environment path for the x64 platform would be the following: **`PATH=$(MATLABROOT)\bin\win64`**

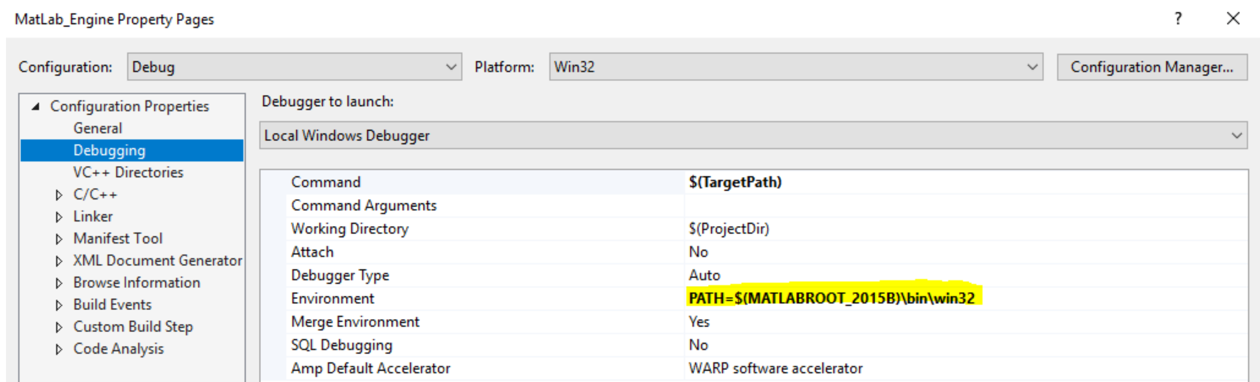

Figure 2: Supplying the Path Variable from within the Project

→ The MatLab API is exposed to your C++ environment via a header file called "engine.h", which resides at a location that y86 our project needs to know. The header file for the x86 platform, targeting the MatLab2015b installation, can be seen in the figure below. The include file for the x64 platform, targeting the MatLab2018b installation is located here: **`$(MATLABROOT)\extern\include`**

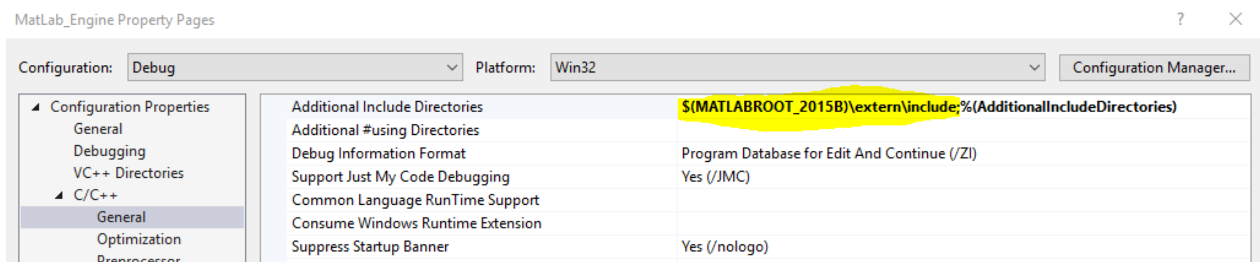

Figure 3: Specifying the Include File Directory

→ As we mentioned before, when instantiating the MatLab engine from C++, several DLLs are loaded to provide the MatLab functionality that we want to exploit using the engine. The three initial DLLs that will be loaded are libmx.dll, libmat.dll, and libeng.dll, and the names and path of the associated export libraries must be known to the project. The names of the export libraries are identical in the x64 and x84 platforms.

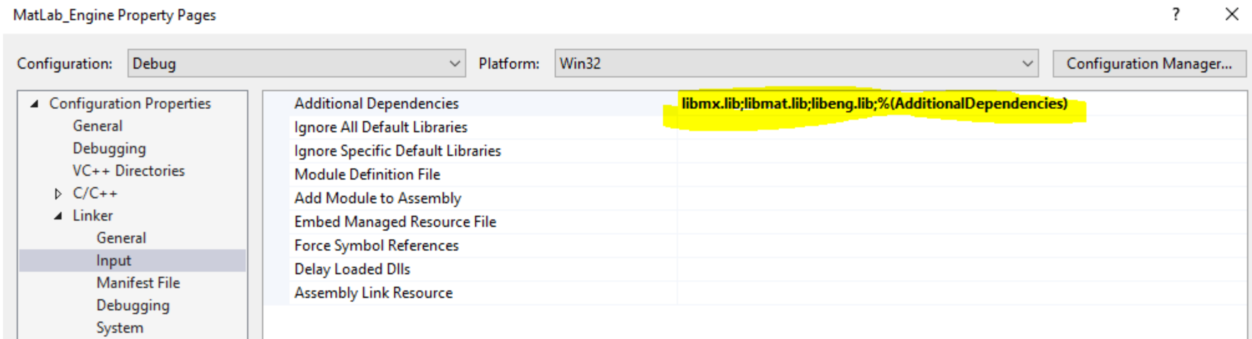

Figure 4: Enumerate the Export Library Name

→ The location of the export libraries for the x64 platform (MatLab 2018b) are at $(MATLABROOT)\extern\lib\win64\microsoft, whereas those for the x32 platform (MatLab 2015b) are shown below.

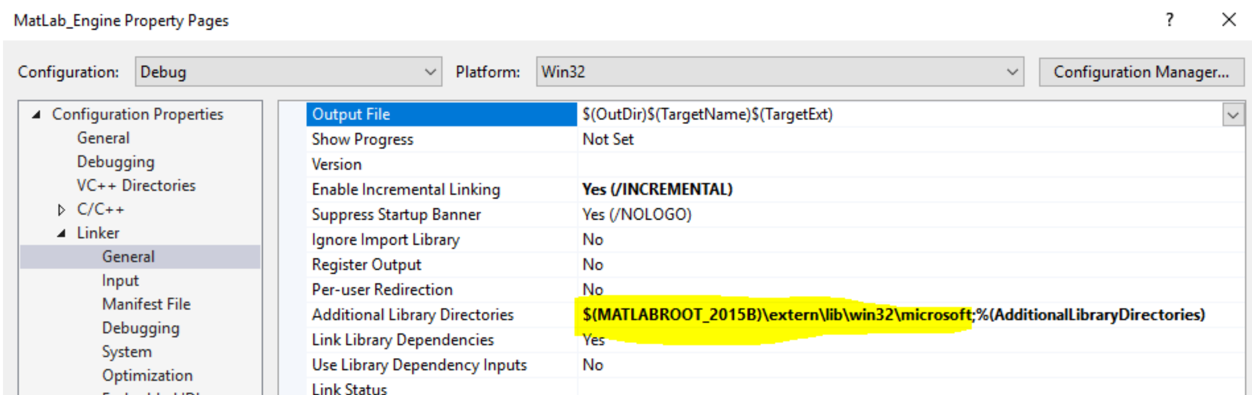

Figure 5: Export Library Location

Running a Simple Test Application

The program below opens an engine object, declared in “engine.h”, links the output buffer to it and then send the command “ver” to the MatLab command window. The answer, the MatLab version and toolboxes, for which you are licensed, are caught by the output buffer and echoed to your applications console.

```
#include <stdio.h>    // for the various print statements
#include <string.h>   // For the strcpy instruction
#include "engine.h"

#define BUFSIZE 1024

int main()
{
   // -------------------------------------------------
   // 1. Some bare bones declarations
   Engine *ep;                      // A pointer to an Engine object, declared in "engine.h"
   char OutputBuffer[BUFSIZE + 1]; // Buffer receives output text from MatLab command window
   OutputBuffer[BUFSIZE] = '\0';    // Properly terminating the char array.
   char InputBuffer[BUFSIZE];       // Text to be sent to the MabLab command window

   // -------------------------------------------------
```

```
   // 2. Call engOpen with a NULL string. This starts a MATLAB process
   //    on the current host using the command "matlab".
   if (!(ep = engOpen("")))
   {
      fprintf(stderr, "Can't start MATLAB engine\n");
      return EXIT_FAILURE;
   }

   //    Attach the output buffer to the engine.
   engOutputBuffer(ep, OutputBuffer, BUFSIZE);

   // --------------------------------------------------
   // 3. Send the command "ver" to the MatLab command window. It will report the name of
   //    the MatLab installation and enumerated the toolboxes for which you have a license.
   strcpy(InputBuffer, "ver");
   engEvalString(ep, InputBuffer);
   printf("%s", OutputBuffer);

   // 4. Close the MatLab engine.
   engClose(ep);

   return EXIT_SUCCESS;
}
```

```
C:\Work\Miscellaneous_Repo\MatLabEngine32_64\x64\Debug\MatLab_Engine.exe
-----------------------------------------------------------------------------------------
MATLAB Version: 8.6.0.267246 (R2015b)
MATLAB License Number: 40760100
Operating System: Microsoft Windows 10 Enterprise Version 10.0 (Build 17134)
Java Version: Java 1.7.0_60-b19 with Oracle Corporation Java HotSpot(TM) Client VM mixed mode
-----------------------------------------------------------------------------------------
MATLAB                                              Version 8.6         (R2015b)
Antenna Toolbox                                     Version 1.1         (R2015b)
Communications System Toolbox                       Version 6.1         (R2015b)
DSP System Toolbox                                  Version 9.1         (R2015b)
Phased Array System Toolbox                         Version 3.1         (R2015b)
RF Toolbox                                          Version 2.17        (R2015b)
Signal Processing Toolbox                           Version 7.1         (R2015b)
```

Figure 6: Visual Studio Console Output in Response to ‘ver’ Command Sent to MatLab Engine

An Example Returning a Structure Containing a Title and a Complex Array

MatLab’s strength is the ease with which it can perform mathematical operations in engineering and science applications. In this examples, we will open the MatLab engine and create a structure with two fields, one containing a character string and the other an array of complex numbers. The example will teach us how to extract a MatLab variable of type mxArray and convert its content into types that are common in C++.

```
% This simple script computes the values of a complex sinusoid and returns the result in a
% structure that contains the values of the sinusoid as well as a descriptive char string.
function Output = GenerateWaveform()

N               = 100;   % Number of samples in waveform
Frequency       = 10;    % Frequency of sinusoid
SampleRate      = 1000;  % Sample rate of the waveform
n               = 0:N-1; % Descrite time index

Signal          = exp(1j*2*pi*n*Frequency/SampleRate);

% The output structure
Output.title    = 'Complex Waveform';
Output.waveform = Signal;
```

The following code executes the following steps.

→ It opens the MatLab engine and attaches the output buffer.
→ It changes the directory to the location of the GenerateWaveform() script a executes it.
→ The MatLab structure 'Output' is imported into C++ as an mxArray.
→ We extract the 'title' field of the mxArray and then print it to the C++ console.
→ We extract the 'waveform' field of the mxArray and then print its content to the C++ console.

```
#include <stdio.h>     // for the various print statements
#include "engine.h"
#include <vector>
#include <complex>

using Complex  = std::complex<float>;
using vComplex = std::vector<Complex>;

#define  BUFSIZE 256

int main()
{
   // ---------------------------------------------------------------------------------------
   // 1. Let's get set up
   Engine *ep;                 // A pointer to an Engine object
   mxArray *pOutput = NULL;    // A pointer to an mxArray object that will hold the output of
                               // our MatLab script
   char OutputBuffer[BUFSIZE + 1];  // The output buffer will receive output text from the
                                    // MatLab command window
   OutputBuffer[BUFSIZE] = '\0';    // Terminate buffer
   char InputBuffer[BUFSIZE];       // The input buffer will contain text to be sent to the
                                    // MabLab command window

   // ---------------------------------------------------------------------------------------
   // 2. Call engOpen with a NULL string. This starts a MATLAB process
   //    on the current host using the command "matlab".
   if (!(ep = engOpen(""))) {
      fprintf(stderr, "Can't start MATLAB engine\n");
      return EXIT_FAILURE;
   }

   // Attach the output buffer to the engine.
   engOutputBuffer(ep, OutputBuffer, BUFSIZE);


   // ---------------------------------------------------------------------------------------
   // 3. Send MatLab a command to change the directory
   strcpy(InputBuffer, "cd c:/work");
   engEvalString(ep, InputBuffer);

   // 4. Send MatLab a command to run the GenerateWaveform.m script.
   strcpy(InputBuffer, "Output = GenerateWaveform()");
   engEvalString(ep, InputBuffer);
   printf("%s", OutputBuffer);


   // ---------------------------------------------------------------------------------------
   // 5. The variable Output is now part of MatLabs work space. We need to retrieve the
   // variable, which is done here. The mxGetClassName() function produces quite a bit of
   // detail regarding the Output variable.
   if ((pOutput = engGetVariable(ep, "Output")) == NULL)
      printf("Oops! You didn't create a variable Output.\n\n");
   else {
      printf("Output is class %s\t\n", mxGetClassName(pOutput));
   }
```

```
   // -------------------------------------------------------------------------------------------
   // 6.  There are two fields to the result. They are called 'title' and 'waveform'
   // 6a. Let's extract the title field of the mxArray, which is pointed to by pOutput.
   mxArray* pTitle          = mxGetField(pOutput, 0, "title");
   char     Title[100];
   mxGetString(pTitle, Title, 100);
   printf("%s %s\n", "The title of the structure is called: ", Title);

   // 6b. Let's extract the waveform field of the mxArray, which is pointed to by pOutput.
   mxArray* pFinalWaveform  = mxGetField(pOutput, 0, "waveform");

   // The array will have one row (technically, we don't even have to call this function
   size_t Rows = mxGetM(pFinalWaveform);

   // The array will have one column (these are the number of entries in the waveform)
   size_t Columns = mxGetN(pFinalWaveform);

   // Create a vector of type Complex, into which we will transfer the complex waveform samples
   // returned by the MatLab script.
   vComplex OutputVector(Columns, Complex(0, 0));

#ifdef MATLAB2018A_AND_LATER
   // MatLab2018a and later version use an interleaved format to store complex arrays and the
   // methodology of gaining access to that array have changed as seen below.
   mxComplexDouble* pc = mxGetComplexDoubles(pFinalWaveform); // Get pointer to interleaved
                                                               // complex array
   for (dword dwIndex = 0; dwIndex < Columns; dwIndex++)
   {
      double dReal = (*pc).real;              // Extracting real component at pointer location
      double dImag = (*pc).imag;              // Extracting imag component at pointer location
      OutputVector[dwIndex] = Complex(dReal, dImag);
      pc++;
   }
#else
   // MatLab2017b and earlier versions stored the real and imaginary components of an mxArray
   // object as seperate arrays that would have to be addressed seperately.
   double* pRealData = mxGetPr(pFinalWaveform); // Get pointer to real part of complex array
   double* pImagData = mxGetPi(pFinalWaveform); // Get pointer to imag part of complex array

   printf("%s\n", "The waveform has the following elements.");
   for (unsigned long dwIndex = 0; dwIndex < Columns; dwIndex++)
   {
      double dReal = *pRealData;
      double dImag = *pImagData;
      OutputVector[dwIndex] = Complex((float)dReal, (float)dImag);
      printf("%s%f%s%f\n", "Real: ", dReal, "   Imag: ", dImag);
      pRealData++;
      pImagData++;
   }
#endif
   engClose(ep);
   return EXIT_SUCCESS;
}
```

The following link provides additional information about how accessing complex arrays of mxArrays has changed in release 2018a.

https://www.mathworks.com/help/matlab/matlab_external/matlab-support-for-interleaved-complex.html