A guide to compiling C++ code to create plugins for DigitalMicrograph (GMS 2.x).

Dave Mitchell

adminnospam@dmscripting.com (remove the nospam to make this address work)

2/10/14

Introduction.

The scripting language in DigitalMicrograph (DM) is interpreted. This means that the user writes code in a high level language which is readily readable by a human being. However, computers do not understand this language, they work in low level binary commands. So when a high level script is run, it must be interpreted into low level commands. This takes time and so interpreted languages are typically executed very slowly. Often times speed is not an issue with DM script. If an image processing operation takes 5s – you wait 5s. However, the fact that DM uses image variables, means that some operations are really fast. For example adding 1 to all the pixels values in a 1k x 1k image (1 million pixels), is near instantaneous, because DM can parallel process images. One way to speed up time-critical scripts is to treat images like data arrays, and store values in them, and then parallel process normal maths operations – but with images instead of variables like x and y. This approach works and good results can be obtained. However, for computationally intensive tasks DM script may simply be too slow.

It is under these circumstances that the heavy lifting can be farmed out to a compiled function. When a piece of code is compiled it is converted into low level language (binary) which the computer can act upon without having to interpret. This can speed things up by several orders of magnitude.

The way it might work is that a DM script is written to do a certain operation. The really computationally intensive part is written as a function in a high level language like C++. This is compiled and converted into a plugin (in binary). The plugin is placed into DM's plugin folder, so that when DM is launched, it reads the plugin code. When the user's script is executed, the script calls the function in the plugin. This does the hard work and returns the result to the script. In this way, the function simply becomes an extension of the DM script instruction set, and it can be called by any script.

In essence it sounds very simple, and in practice it sort of is, but only once you have worked out all the pitfalls – and there are many. This guide is really just a memory dump of my recent experiences with getting a plugin compiled for GMS 2. I had done this once before for GMS 1, and it was a painful experience. This time it was easier, but still involved lots of trial and even more error. This note is designed simply to help ease you over the initial hump of getting a plugin compiled and working. I am not a C++ programmer, and can offer no assistance with your plugin development. I have no doubt I will do more of this stuff in the future, so this may be of as much assistance to me as it is to you.

What you need.

Microsoft Visual Studio 2008 This is the development environment. I cannot comment on whether other environments will work. All I can say is that this one does.

Gatan Software Developers Kit (DMSDK): Gatan have kindly put together a useful SDK which contains examples of the various things you might like to create. The Source code in each plugin is well annotated. However, there is no overarching 'How To', so perhaps this might help fill the gap. Since GMS 2 is available in two flavours: 32 bit and 64 bit, there are SDKs for each environment.

You will need to get a license for two SDKs from Gatan. These are free, simply contact Gatan via email and request the licenses. Note the links on the Gatan website to the SDK are obsolete. Neither of the SDKs listed are compatible with GMS 2. All you need do is email Gatan (http://www.gatan.com/resources/scripting/sdk/) and they will send you the licenses. For installation of the SDKs – see later.

DigitalMicrograph GMS 2.x. This is now a freely available download from Gatan (http://www.gatan.com/resources/DM_License_Request/index.php). In order to test 32 and 64 bit plugins you will need 32 and 64 bit setups of DigitalMicrograph. You cannot have two installations on the same PC – so it is really useful to have either two separate PCs or use virtualization software like Parallels, so you can run an operating system (eg XP) inside another (Mac OSX, Windows 7 etc). If you have a modern Windows 7 PC, install the 64 bit GMS on that and relegate the 32 bit DM installation to the antique XP laptop of whatever you have to hand. Note, there is no difference in functionality between the 32 and 64 bit DM installations – in terms of menu items etc. The only difference is that the 64 bit version can handle very large (>2GB) data sets, like large tomograms or other 3D datasets.

GMS 2.x Installer: This is the same installer from which you installed your DM version. The installer contains all the Gatan software, including the SDKs. What gets installed depends upon which licenses the installer finds. Once you get the SDK licenses from Gatan, you will use this installer to install the SDK.

Boost 1.38: This is a C++ library (free), which the SDK uses. It can be sourced from : http://www.boost.org/users/history/version_1_38_0.html

The SDK says use Boost 1.38 or later. I tried using the latest version (1.55 at the time of writing). It did not work. Make sure you download and use 1.38.

Setting up your software.

Visual Studio 2008 (VS2008). This is pretty straightforward. Simply run the installer on the DVD and follow your nose. The only really exasperating gotcha concerns the compiler for 64 bit. If you do a basic install of VS2008, the 64 bit compiler does not get installed, only the 32 bit compiler does. If you attempt to compile a 64 bit plugin without the 64 bit compiler installed, does the software say ”I am sorry Dave, I can’t do that, you haven’t installed the 64 bit compiler”. No, it doesn’t. Instead it simply skips the compile job. No errors, no prompts, nothing. So, now you know. When you install VS2008, use the FULL install, even if you have no immediate intentions of working with 64 bit. Having discovered that from trawling various discussion groups, the advice was to modify the existing installation using the Programs and Features control panel (Windows 7). However, that didn’t work for me and the only option was to do a full uninstall. Sounds simple – just run the uninstall option from the Programs and Features control panel (Windows 7). That didn’t work either. Further web trawling indicated that Microsoft have a stand-alone program for doing this – do a Google search on ‘VS 2008 uninstall tool’. Once VS 2008 has been uninstalled I could then reinstall it, making sure I used the full version. There is an option to install the help documentation. However, I find Microsoft help documents opaque at the best of time and circularly referencing most of the time. In VS 2008 this is taken to whole new level of futility – install if you wish.

DigitalMicrograph. Everything described in this document relates to GMS 2.x. If you are using GMS 1.x, none of this applies. Run the license installer, then run the GMS installer. If you have a 64 bit computer, you can install either the 32 bit or the 64 bit version, though I’d choose the latter. On 32 bit PCs, only the 32 bit GMS can be installed.

SDK: Once you have sourced the SDK licenses from Gatan, copy the SDK licenses relevant to your GMS (32 or 64 bit – but not both) to the folder in which your current GMS licenses are stored. These can be found at:

Windows 7: C:\ProgramData\Gatan\Licenses. Note the ProgramData folder is normally hidden. From the Control Panel select Folder Options – click on the View tab and select Show Hidden Files, Folders.

XP: C:\Documents and Settings\All Users\Application Data\Gatan\Licenses

Run the GMS installer again, and this time you should see the SDK selected automatically by the installer. Just follow your nose and let the installer install the SDK. When it has finished, you should have new folder called DMSDK in the following location (Windows 7) C:\ProgramData\Gatan. Note the ProgramData folder is normally hidden, so set the Show Hidden Files, Folders option as described above. Failing that, do a search on DMSDK.

The DMSDK folder can be moved onto the desktop. In fact, it is a good idea to keep a virgin copy of this DMSDK folder safe and work on a copy. That way, if you really mess up something while experimenting, you can trash the whole folder, make a fresh copy of the virgin folder to try again. Another reason to work with a copy on your desktop or similar, is to ensure you have the necessary privileges to write to the folder.

There is nothing to differentiate the 32 from the 64 bit DMSDK folder in terms of labeling. However, you can tell them apart if you look inside the Lib folder therein. This will contain a folder called x64 if the DMSDK is the 64 bit version and Win32 if the DMSDK is the 32 bit version.

The DMSDK contains examples of lots of different things – few of which I pretend to understand. The folder I have worked with exclusively is called Library Example. This shows how to create a dynamic linked library (dll). This is compiled binary plugin which lives in the Gatan Plugins folder and which extends the DM scripting language to include whatever commands are encoded therein. All further discussion relates to getting this Library Example to compile. I have not worked with any of the other examples and can offer no guidance on their implementation.

Boost 1.38. You should have downloaded this folder of library files and decompressed it from whatever compressed format was used. It is a good idea to build a repository for all your installers, folders, DMSDKs, licenses etc and label them carefully. Store this somewhere securely and always work with copies. This will make repairing or replacing trashed files easy. Good labeling is very important, as licenses for DM and the SDK may look similar and the various SDKs are easily confused.

Drag a copy of the Boost_1_38 folder into the DMSDK folder on your desktop. You will need to link your Visual Studio (VS) 2008 project to this folder (see later), so the relevant library files can be included in the plugin.

Your software environment is now setup. It should be straightforward, but as you can see, there are pitfalls which will cost you time – hopefully less so, now that you have read this.

Using VS 2008 to build a dll

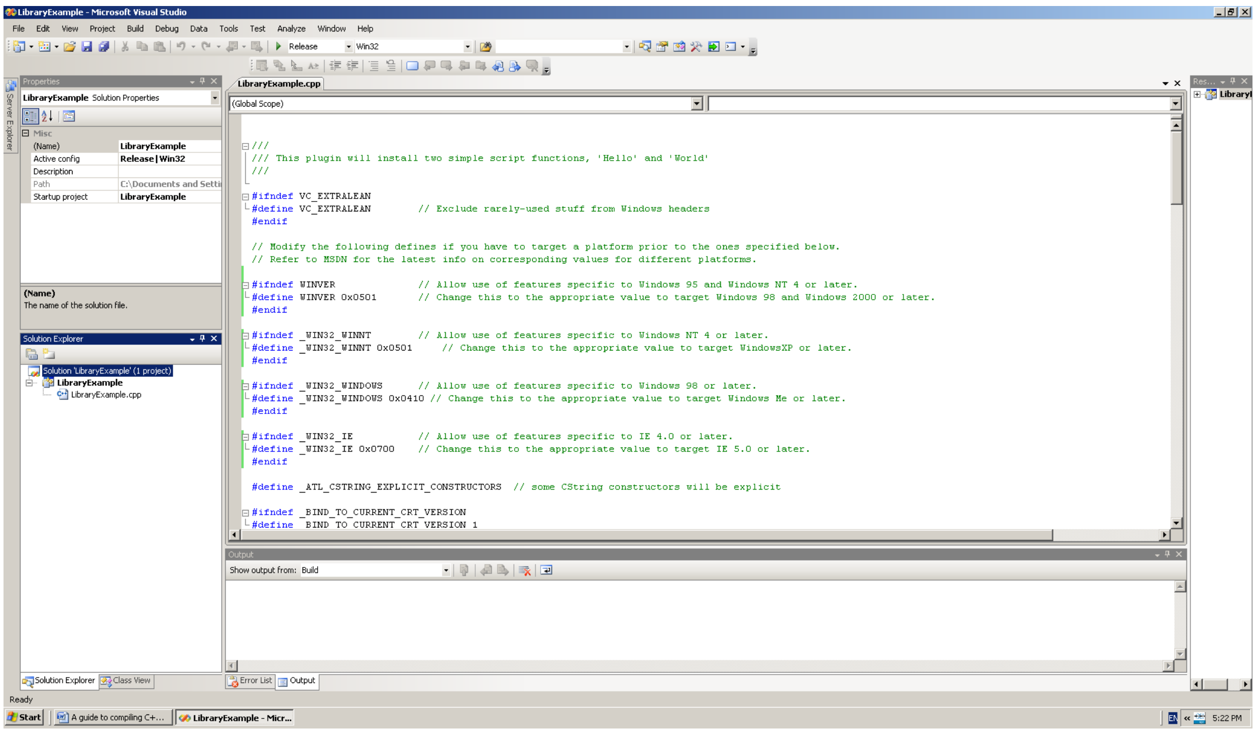
Opening and configuring a project

When you build your projects you will encounter many weird and wonderful errors which may mean nothing at all to you. You can look them up on the web or in the help – but generally it is hard to make sense of a lot of the stuff in there. The best approach is to feel your way very tentatively, making small incremental changes to the code with frequent checking to see if it will compile. If it does not, at least you know which bit of code is wonky. If you add lots of code and there are multiple issues, the compilation errors and warning will compound and you may be hard pressed to work out what is wrong. Make frequent of copies of your project as you edit your C++ code, that way you can always backtrack if things go awry and you break your code too badly to fix it. Annotate your code very extensively. This will not only help you troubleshoot but will enable to retrace your steps if you revisit your code in a week or a year's time.

Launch VS 2008. You can close the Start page by clicking in the top right corner of the start page – not the top right corner of the application window.

From the File menu select Open Project/Solution.

Assuming the DMSDK (a copy not the original) is on your desktop navigate to it and open the Library Example folder and double click on the Library Example project file.


LibraryExample - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Analyze Window Help
Release
Win32
Properties
Solution Explorer
Solution 'LibraryExample' (1 project)
LibraryExample
LibraryExample.cpp
LibraryExample.cpp
(Global Scope)
Output
Error List
Output
Ready
Start

LibraryExample Solution Properties

(Name)	LibraryExample
Active config	Release | Win32
Description	
Path	C:\Documents and Setti
Startup project	LibraryExample

(Name)
The name of the solution file.

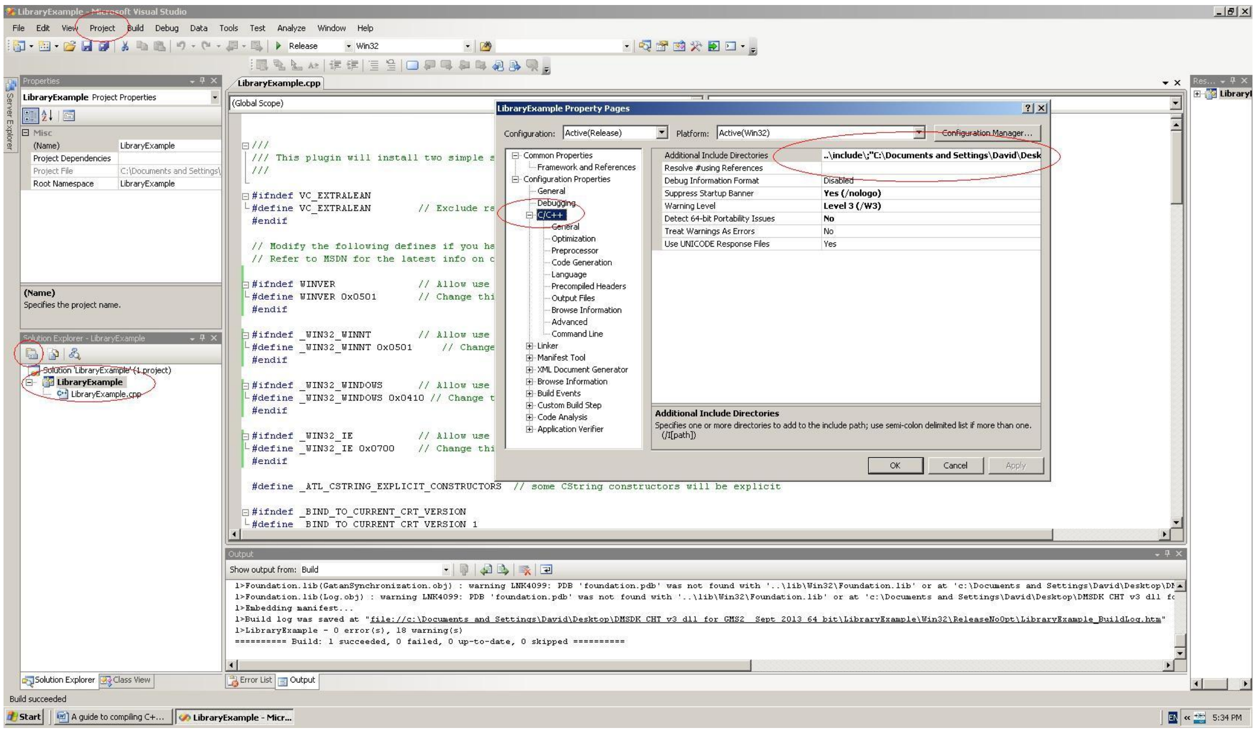
```
///
/// This plugin will install two simple script functions, 'Hello' and 'World'
///

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers
#endif

// Modify the following defines if you have to target a platform prior to the ones specified below.
// Refer to MSDN for the latest info on corresponding values for different platforms.

#ifndef WINVER              // Allow use of features specific to Windows 95 and Windows NT 4 or later.
#define WINVER 0x0501       // Change this to the appropriate value to target Windows 98 and Windows 2000 or later.
#endif

#ifndef _WIN32_WINNT        // Allow use of features specific to Windows NT 4 or later.
#define _WIN32_WINNT 0x0501     // Change this to the appropriate value to target WindowsXP or later.
#endif

#ifndef _WIN32_WINDOWS      // Allow use of features specific to Windows 98 or later.
#define _WIN32_WINDOWS 0x0410 // Change this to the appropriate value to target Windows Me or later.
#endif

#ifndef _WIN32_IE           // Allow use of features specific to IE 4.0 or later.
#define _WIN32_IE 0x0700    // Change this to the appropriate value to target IE 5.0 or later.
#endif

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS  // some CString constructors will be explicit

#ifndef _BIND_TO_CURRENT_CRT_VERSION
#define _BIND_TO_CURRENT_CRT_VERSION 1
```

Show output from: Build

Screen shot of the loaded Library Example project from the 32 bit DMSDK

A few things to notice. The actual code for the plugin is the LibraryExample.cpp file. Clicking on this will display the code in the main window – where editing can be carried out. The 32 bit vs 64 bit selection can be made at the top of the screen – just below the Help menu. Note when you compile this code, the contents of the Library folder from the DMSDK folder is included in the build. You can only compile a 32 bit plugin when this library file is 32 bit ie you are using the 32 bit version of the DMSDK. Similarly, you can only select the x64 (64 bit) option when using the 64 bit version of the DMSDK. Actually you can select the 64 bit option – but it will not compile, unless the correct Library file (64 bit) is present.

Another option is the Release and ReleaseNoOpt selection – next to the 32/64 bit selection. I do not know what the difference between these two is – I always used Release.



LibraryExample Property Pages

Configuration: Active(Release) Platform: Active(Win32)

Additional Include Directories	..\include\;"C:\Documents and Settings\David\Desk
Resolve #using References	
Debug Information Format	Disabled
Suppress Startup Banner	**Yes (/nologo)**
Warning Level	**Level 3 (/W3)**
Detect 64-bit Portability Issues	**No**
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

(Name)	LibraryExample
Project Dependencies	
Project File	C:\Documents and Settings\
Root Namespace	LibraryExample

```
///
/// This plugin will install two simple s
///

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN        // Exclude ra
#endif

// Modify the following defines if you ha
// Refer to MSDN for the latest info on c

#ifndef WINVER              // Allow use
#define WINVER 0x0501       // Change thi
#endif

#ifndef _WIN32_WINNT        // Allow use
#define _WIN32_WINNT 0x0501     // Change
#endif

#ifndef _WIN32_WINDOWS      // Allow use
#define _WIN32_WINDOWS 0x0410 // Change t
#endif

#ifndef _WIN32_IE           // Allow use
#define _WIN32_IE 0x0700    // Change thi
#endif

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS  // some CString constructors will be explicit

#ifndef _BIND_TO_CURRENT_CRT_VERSION
#define _BIND_TO_CURRENT_CRT_VERSION 1
```

```
1>Foundation.lib(GatanSynchronization.obj) : warning LNK4099: PDB 'foundation.pdb' was not found with '..\lib\Win32\Foundation.lib' or at 'c:\Documents and Settings\David\Desktop\DM
1>Foundation.lib(Log.obj) : warning LNK4099: PDB 'foundation.pdb' was not found with '..\lib\Win32\Foundation.lib' or at 'c:\Documents and Settings\David\Desktop\DMSDK CHT v3 dll fo
1>Embedding manifest...
1>Build log was saved at "file://c:\Documents and Settings\David\Desktop\DMSDK CHT v3 dll for GMS2  Sept 2013 64 bit\LibraryExample\Win32\ReleaseNoOpt\LibraryExample_BuildLog.htm"
1>LibraryExample - 0 error(s), 18 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Build succeeded

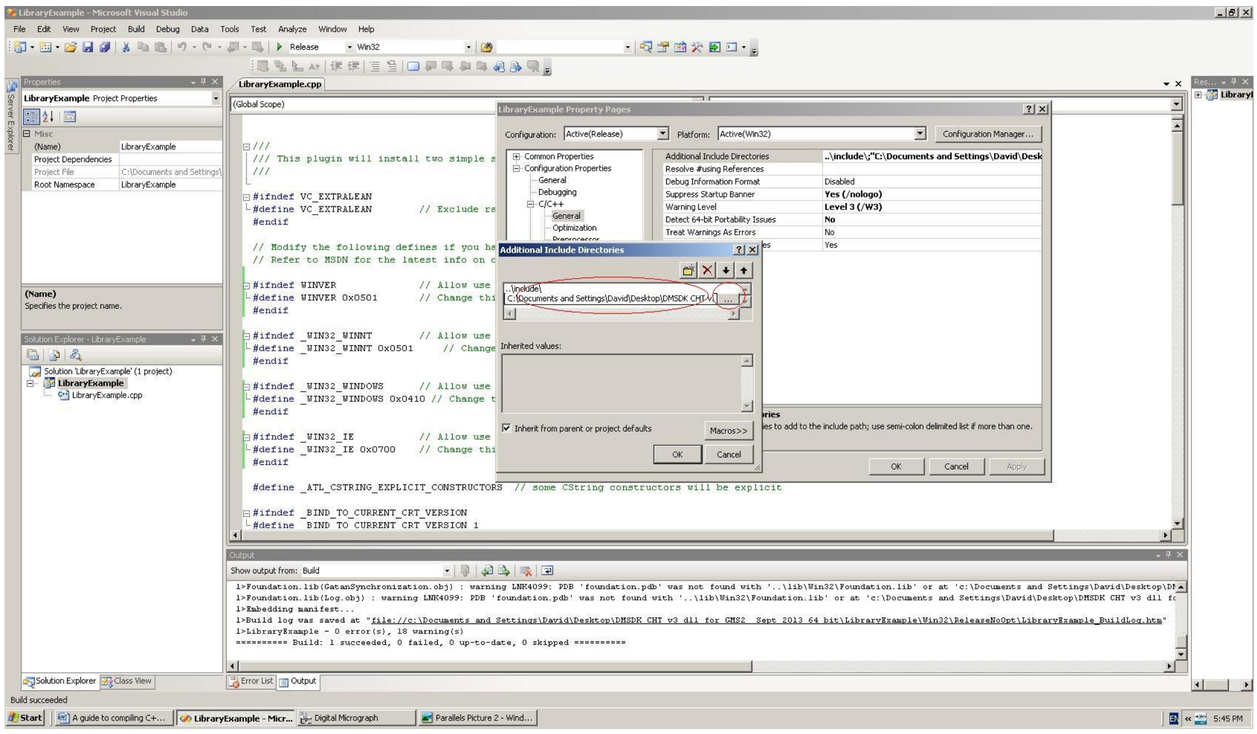
Screenshot I showing how to set the correct path so that the Boost library files can be accessed by the LibraryExample project.

Previously you should have downloaded the Boost 1.38 library and copied this folder inside the DMSDK folder on your desktop. In order for the project you have opened (LibraryExample) to see that Boost folder, you have to point at it.

Click ONCE on the LibraryExample project icon in the Solution Explorer window in the bottom left of the screen to select it. You can either click on the small icon in the Solution Explorer title bar (ringed in red), or from the Projects menu select Properties. The Project Property Pages dialog opens.

Click ONCE on the C/C++ tab to highlight it and display the properties relating to C/C++ files will be displayed.

Double click in the row showing the paths to the Include directories (ringed in red). This is immediately to the right of the Additional Include Directories label. You'll notice an inconspicuous button appear at the end of the row. Press this button and a dialog to identify the path to the Boost directory will appear.



```
///
/// This plugin will install two simple s
///

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN        // Exclude ra
#endif

// Modify the following defines if you he
// Refer to MSDN for the latest info on c

#ifndef WINVER              // Allow use
#define WINVER 0x0501       // Change thi
#endif

#ifndef _WIN32_WINNT        // Allow use
#define _WIN32_WINNT 0x0501    // Change
#endif

#ifndef _WIN32_WINDOWS      // Allow use
#define _WIN32_WINDOWS 0x0410 // Change t
#endif

#ifndef _WIN32_IE           // Allow use
#define _WIN32_IE 0x0700    // Change thi
#endif

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS  // some CString constructors will be explicit

#ifndef _BIND_TO_CURRENT_CRT_VERSION
#define _BIND_TO_CURRENT_CRT_VERSION 1
```

LibraryExample Property Pages — Configuration: Active(Release), Platform: Active(Win32)

Additional Include Directories	..\include\;"C:\Documents and Settings\David\Desk
Resolve #using References	
Debug Information Format	Disabled
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level 3 (/W3)
Detect 64-bit Portability Issues	No
Treat Warnings As Errors	No
es	Yes

Additional Include Directories:

```
..\include\
C:\Documents and Settings\David\Desktop\DMSDK CHT v
```

Inherit from parent or project defaults [✓]

```
1>Foundation.lib(GatanSynchronization.obj) : warning LNK4099: PDB 'foundation.pdb' was not found with '..\lib\Win32\Foundation.lib' or at 'c:\Documents and Settings\David\Desktop\DM
1>Foundation.lib(Log.obj) : warning LNK4099: PDB 'foundation.pdb' was not found with '..\lib\Win32\Foundation.lib' or at 'c:\Documents and Settings\David\Desktop\DMSDK CHT v3 dll fc
1>Embedding manifest...
1>Build log was saved at "file://c:\Documents and Settings\David\Desktop\DMSDK CHT v3 dll for GMS2  Sept 2013 64 bit\LibraryExample\Win32\ReleaseNoOpt\LibraryExample_BuildLog.htm"
1>LibraryExample - 0 error(s), 18 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Screenshot II showing how to set the correct path so that the Boost library files can be accessed by the LibraryExample project.

The DMSDK has a folder called Include. This contains code which gets included with the plugin. The first include directory listed points to this folder – do not change it.

If there is an existing path to the Boost 1.38 directory, then it will be incorrect – since you will not have placed Boost in the same location as the Gatan software engineer who wrote this example. If an existing Boost path is present change it. If not, create a new one. Do this by clicking on the path line to make another small button appear at the end of the row (ringed in red). Click on this and identity the Boost directory. Point at the Boost_1_38 parent folder which contains several folders and files. Do NOT point at the Boost folder which lives inside the Boost_1_38 parent folder.

If you have not set this path correctly, then when you attempt to build the project an error will occur, telling you that Boost 1.38 cannot be found.

Everything is now ready for compiling (building) the solution.

Building a solution

Note when you carry out a build, the resulting files are output into a solution folder. It is a good idea to clean the Solution before testing some new addition or change. This deletes all the files created in the previous build. The sequence is Build – Clean – Build – Clean . . . and repeat until you have a working solution and the project will compile correctly.

To clean a Solution from the Build menu select Clean Solution.

The Output window at the bottom of the VS 2008 screen will give you a running update on cleaning and building progress. Monitor this window for alerts, warnings and errors.

To build the Project, from Build menu select Build Solution. The LibraryExample takes about 5s to compile. What you want to appear in the Output window at the end of a build is:

```
1>------ Build started: Project: LibraryExample, Configuration: Release Win32 ------
1>Compiling...
1>LibraryExample.cpp
1>Linking...
1>   Creating library c:\Documents and Settings\David\Desktop\DMSDK CHT v3 dll for GMS2  Sept
2013 64 bit\LibraryExample\Win32\Release/LibraryExample.lib and object c:\Documents and
Settings\David\Desktop\DMSDK CHT v3 dll for GMS2  Sept 2013 64
bit\LibraryExample\Win32\Release/LibraryExample.exp
1>Embedding manifest...
1>Build log was saved at "file://c:\Documents and Settings\David\Desktop\DMSDK CHT v3 dll for
GMS2  Sept 2013 64 bit\LibraryExample\Win32\Release\LibraryExample_BuildLog.htm"
1>LibraryExample - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

This indicates that compilation was successful and you now have a plugin to test.

You will find the compiled plugin at the following location:

Desktop/DMSDK/LibraryExample/Win32/Release/LibraryExample.dll

This assumes that the DMSDK was on your desktop and that you had the project configured for 32 bit Release. If you used 32 bit ReleaseNoOpt, then look in the corresponding ReleaseNoOpt folder. If you compile for 64 bit then instead of a Win32 folder there will be an x64 folder.

In the case of the LibraryExample we are compiling a dll and the file we are interested in is called LibraryExample.dll. The Solution contains a number of files with the same name. List the folder contents by Details and select the file with the .dll extension.

Testing the plugin

To test this plugin, ensure that DigitalMicrograph is not running. You will need to copy the LibraryExample.dll file into DigitalMicrograph's plugin folder. The location will vary with your operating system:

GMS 1.x plugin (Windows XP 32bit): C:\Program Files\Gatan\DigitalMicrograph\PlugIns

GMS 2.x plugin 32 bit (Windows 7 32 or 64bit): C:\Program Files\Gatan\Plugins

GMS 2.x plugin 64 bit (Windows 7 64bit): C:\Program Files\Gatan\Plugins

The first test is to launch DigitalMicrograph. You should see plugins flashing up on the start up panel as they load. If your plugin loads without error – that's a good start.

Test your plugin. Typically you will be creating additional functions for the DM scripting language. Write a short DigitalMicrograph script to call your new function and make sure the output is what you expect.

To test the LibraryExample.dll plugin the following short script will work.

```
// Script to test the LibraryExample.dll plugin provided in the DMSDK

// test of Hello() function

result("\nOutput from hello() function:\n")

hello() // First function added by the LibraryExample.dll


// Test of World() function

result("\n\nOutput from world() function:\n")

world() // Second function added by the LibraryExample.dll
```

If the dll is working correctly, calling the hello() function outputs 'Hello,' to DigitalMicrograph's output window, while calling the world() function outputs 'Hello World . . .'

The approach I used for developing my CircularHoughTransform.dll plugin (http://www.dmscripting.com/chtdiffractionanalysis.html) was to edit the LibraryExample.cpp C++ code in VS 2008 to remove one of the functions and rename the other to the name of my function. I then went through the rest of the code and modified it so it added my function to DM. I then test compiled it to see if all was OK. Only then did I start actually modifying the body of the function. After each change, I used Build Solution to make sure the code was compiling OK, then moved on to the next change. I kept whittling away at it until I had a working solution.

I hope this guide helps. I am not a C++ programmer, and what you see here is the sum total of my knowledge. I cannot offer any meaningful assistance with C++ code development. There are more knowledgeable folk out there in the DM scripting community – including several who do this sort of thing for a living.

Note HREM Research (http://www.hremresearch.com) is a commercial company which develops DM scripts and plugins and markets code commercially for small developers. If you have a script package with commercial potential which needs some C++ development or marketing they may be able to assist.

If you find anything in here that is incorrect – please let me know.

Kind regards,

Dave Mitchell