CiaransLabBook

Release v0.0.1

Jan 29, 2021

Contents

sphinx-quickstart on Tue Jul 28 12:47:19 2020. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

I keep looking up the same commands over and over again, so here I'll denote some useful ones.

CHAPTER 1

Docker

1.1 Commands

Turn off all containers

```
docker container stop $(docker container ls -aq)
```

using a filter

```
docker container prune --filter "until=12h"
```

A great resource on explaining the docker basics

Docker build

```
DOCKER_BUILDKIT=1 docker build -t ciaranwelsh/libomexmeta-build:latest .
```

Push to dockerhub

```
docker push ciaranwelsh/libomexmeta-build:latest
```

CHAPTER 2

CMake

2.1 Cross platform CMake

https://gitlab.kitware.com/cmake/community/-/wikis/doc/tutorials/How-To-Write-Platform-Checks

Note: cmake is now on pip version 3.17. pip install cmake.

2.2 Copy or install a file

Copy during configuration stage

```
file(COPY ${LIBXML2_LIBRARY} DESTINATION ${PYSEMSIM_DIR})
```

Copy at install time

```
install(FILES ${LIBXML2_LIBRARY}
DESTINATION ${PYSEMSIM_DIR})
```

2.3 API Control

We should consider both what IS in our API and what isn't. Public header files are okay, but its possible for develops to still use things you don't want them to. Instead we can use symbol visibility. Heres a class

```
class MyGenerator {
public:
    int nextValue();
};
```

With visual studio DLLs, this class would be hidden by default. However, on GCC and Clang, this class is visible by default.

On visual studio _declspec you use __*declspec(export)* to change visibility from hidden to visible.

Watch this video: https://www.youtube.com/watch?v=m0DwB4OvDXk And make notes here!.

CHAPTER 3

Git

3.1 Submodule

Update submodules

```
git submodule update --init --recursive
```

CHAPTER 4

Linux

4.1 Find a library on the system

There seems to be multiple ways to do this, and sometimes one command works over another, not sure why.

```
$ ldconfig -p | grep "name-of-lib"
```

```
$ dpkg -L "name-of-lib"
```

Requires installing apt-file

```
$ apt-file search "name-of-lib
```

ldd - print shared object dependencies. Very useful for debugging missing shared libraries.

```
$ ldd $(which curl)
```

Can also try grep with ls -R

```
$ ls -R | grep file
```

Then there is find

```
$ find . -name "*sql*"
```

CHAPTER 5

Building on linux

5.1 Linking static libraries into shared

When passing arguments to the linker you need to ensure you use the *-Wl,–whole-archive* and *-Wl,–no-whole-archive* option. Wrap these around static libraries that you are tyring to pull into a shared library.

```
-Wl,--whole-archive
-lxml2
-Wl,--no-whole-archive
```

This is necessary to tell the linker to pull all the functions from the library into the shared library you are building. Otherwise, only some will be pulled in and you will get a linker error.

It seems there is also another way here

Use -l: instead of -l. For example -l:libXYZ.a to link with libXYZ.a. Notice the lib written out, as opposed to -lXYZ which would auto expand to libXYZ.

Note, these commands can be embedded into a CMake script by passing to *TARGET_LINK_LIBRARIES*

```
TARGET_LINK_LIBRARIES(target SHARED -W,l--whole-archive l:xml2 -Wl,no-whole-archive)
```

5.2 Inspecting broken builds

List all the shared object libraries that libx depends on

```
ldd libx.so
```

List the symbols in a library, along with their status (found, undefined etc.)

```
nm libx.so
```

Use the -D option to inspect dynamic symbols only

```
nm libx.so
```

Pipe output of nm into grep to search for specific function

```
nm libx.so | grep somefunction
```

You can examine the Rpath on Linux thus:

```
readelf -d libsemsim.so
```

CHAPTER 6

Windows

6.1 What is the difference between msys and mingw?

Shamelessly stolen from

MinGW doesn't provide a linux-like environment, that is MSYS(2) and/or Cygwin

Cygwin is an attempt to create a complete UNIX/POSIX environment on Windows. MinGW is a C/C++ compiler suite which allows you to create Windows executables - you only need the normal MSVC runtimes, which are part of any normal Microsoft Windows installation.

MinGW provides headers and libraries so that GCC (a compiler suite, not just a "unix/linux compiler") can be built and used against the Windows C runtime.

MSYS is a fork of Cygwin (msys.dll is a fork of cygwin.dll) cygwyn gcc + cygwin environment defaults to producing binaries linked to the (GPL) cygwin dll (or cygwin1.dll???) mingw + msys defaults to producing binaries linked to the platform C lib.

MinGW: It does not have a Unix emulation layer like Cygwin, but as a result your application needs to specifically be programmed to be able to run in Windows,

MinGW forked from version 1.3.3 of Cygwin

Unlike Cygwin, MinGW does not require a compatibility layer DLL and thus programs do not need to be distributed with source code.

This means, other than Cygwin, MinGW does not attempt to offer a complete POSIX layer on top of Windows, but on the other hand it does not require you to link with a special compatibility library.

Cygwin comes with the MingW libaries and headers and you can compile without linking to the cygwin1.dll by using -mno-cygwin flag with gcc. I greatly prefer this to using plain MingW and MSYS. (This does not work any more with cygwin 1.7.6. gcc: The -mno-cygwin flag has been removed; use a mingw-targeted cross-compiler.)

MSYS is a collection of GNU utilities such as bash, make, gawk and grep to allow building of applications and programs which depend on traditionally UNIX tools to be present. It is intended to supplement MinGW and the deficiencies of the cmd shell.

An example would be building a library that uses the autotools build system. Users will typically run "./configure" then "make" to build it. The configure shell script requires a shell script interpreter which is not present on Windows systems, but provided by MSYS.

A common misunderstanding is MSYS is "UNIX on Windows", MSYS by itself does not contain a compiler or a C library, therefore does not give the ability to magically port UNIX programs over to Windows nor does it provide any UNIX specific functionality like case-sensitive filenames. Users looking for such functionality should look to Cygwin or Microsoft's Interix instead.

MSYS2 uses Pacman (of Arch Linux) to manage its packages and comes with three different package repositories: - msys2: Containing MSYS2-dependent software - mingw64: Containing 64-bit native Windows software (compiled with mingw-w64 x86_64 toolchain) - mingw32: Containing 32-bit native Windows software (compiled with mingw-w64 i686 toolchain)

Cygwin provides a runtime library called cygwin1.dll that provides the POSIX compatibility layer where necessary. The MSYS2 variant of this library is called msys-2.0.dll and includes the following changes to support using native Windows programs: 1) Automatic path mangling of command line arguments and environment variables to Windows form on the fly.

MSYS is a fork of an old Cygwin version with a number of tweaks aimed at improved Windows integration, whereby the automatic POSIX path translation when invoking native Windows programs is arguably the most significant.

6.2 DLLs

Lots of information here is from watching a lecture on YouTube

6.2.1 Explicit Linking

Creating a DLL and loading functions from it

Here's a little library that can be compiled as a dll:

```
// Hello.cpp
extern "C" char const * __cdecl GetGreeting()
    {
        return "Hello, C++ Programmers!";
    }
```

You can compile this using visual studio developer command prompt. The /c flag tells cl only to compile and not also link Hello.cpp

```
> cl.exe /c Hello.cpp
```

We have just created Hello.obj. Now we can link into a dll:

```
> link.exe Hello.obj /DLL /NOENTRY /EXPORT:GetGreeting
```

The DLL flag specifies to create a DLL. The NOENTRY flag tells the linker that the dll does not have an entry point and the /EXPORT:GetGreeting tells the linker which functions from the DLL are going to be exported into another library.

Now, since this is a dll, we need another program, the client program to load *GetGreeting()* and use it.

```
// PrintGreeting.cpp
#include <stdio.h>
#include <Windows.h>

int main(){
    HMODULE const HelloDll = LoadLibraryExW(L"test.dll", nullptr, 0);

    /*
     * GetGreetingType is a function pointer for the type we want to load from Hello.
↪dll
     */
    using GetGreetingType = char const* (__cdecl*)();

    // then we load get greeting, casting to the type we loaded.
    GetGreetingType const GetGreeting = reinterpret_cast<GetGreetingType>(
        GetProcAddress(
            HelloDll, "GetGreeting"));

    puts(GetGreeting());

    FreeLibrary(HelloDll);
}
```

We can compile, link and run this program:

```
cl PrintGreeting.cpp
.\PrintGreeting.exe
```

Which prints out:

Using dumpbin.exe

Dumpbin is a program for parsing windows binaries. Note, on windows you can you "/" or "-" to indicate that what follows is an option. Additionally, the commands are case insensitive.

There are a bunch of headers or metadata inside the dll that can be interrogated using:

DLL Headers

```
dumpbin /HEADERS Hello.
```

DLLs have a predefined structure. First, a bunch of header sections followed by a number of sections, which contain actual code, data and resources in the dll.

The section headers told us where to find the data in the file. We can look at whats actually inside of a section using the *-rawdata* flag.

DLL Raw data

```
dumpbin -rawdata -section:.text Hello.dll
```

So it contains some bytes. We can also disassemble the bytes:

Disassembley

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /disasm -section:.text Hello.dll
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file Hello.dll

File Type: DLL

SECTION HEADER #1
   .text name
       A virtual size
    1000 virtual address (10001000 to 10001009)
     200 size of raw data
     400 file pointer to raw data (00000400 to 000005FF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
60000020 flags
         Code
         Execute Read

  10001000: 55                 push        ebp
  10001001: 8B EC              mov         ebp,esp
  10001003: B8 00 20 00 10     mov         eax,10002000h
  10001008: 5D                 pop         ebp
  10001009: C3                 ret

  Summary

        1000 .text
```

RData

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /rawdata -section:.rdata test.dll
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file test.dll

File Type: DLL

SECTION HEADER #2
  .rdata name
      D8 virtual size
    2000 virtual address (10002000 to 100020D7)
     200 size of raw data
     600 file pointer to raw data (00000600 to 000007FF)
       0 file pointer to relocation table
       0 file pointer to line numbers
```

(continues on next page)

(continued from previous page)

```
       0 number of relocations
       0 number of line numbers
40000040 flags
         Initialized Data
         Read Only

RAW DATA #2
  10002000: 48 65 6C 6C 6F 2C 20 43 2B 2B 20 50 72 6F 67 72  Hello, C++ Progr
  10002010: 61 6D 6D 65 72 73 21 00 00 00 00 00 3B 0A 20 5F  ammers!.....;. _
  10002020: 00 00 00 00 0D 00 00 00 50 00 00 00 88 20 00 00  ........P.... ..
  10002030: 88 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  10002040: 00 00 00 00 FF FF FF FF 00 00 00 00 72 20 00 00  ....ÿÿÿÿ....r ..
  10002050: 01 00 00 00 01 00 00 00 01 00 00 00 68 20 00 00  ............h ..
  10002060: 6C 20 00 00 70 20 00 00 00 10 00 00 7B 20 00 00  l ..p ......{ ..
  10002070: 00 00 74 65 73 74 2E 64 6C 6C 00 47 65 74 47 72  ..test.dll.GetGr
  10002080: 65 65 74 69 6E 67 00 00 00 00 00 00 00 10 00 00  eeting..........
  10002090: 0A 00 00 00 2E 74 65 78 74 24 6D 6E 00 00 00 00  .....text$mn....
  100020A0: 00 20 00 00 40 00 00 00 2E 72 64 61 74 61 00 00  . ..@....rdata..
  100020B0: 40 20 00 00 48 00 00 00 2E 65 64 61 74 61 00 00  @ ..H....edata..
  100020C0: 88 20 00 00 50 00 00 00 2E 72 64 61 74 61 24 7A  . ..P....rdata$z
  100020D0: 7A 7A 64 62 67 00 00 00                          zzdbg...

  Summary

        1000 .rdata
```

Note that we can see where our string is stored. Moreover, the locations of the Export and Debug directories are also located in here.

DLL Exports

The export directory defines the public service of the dll, all the things that other dlls or exes can use from this dll. We can look at these with:

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin -exports test.dll
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file test.dll

File Type: DLL

  Section contains the following exports for test.dll

    00000000 characteristics
    FFFFFFFF time date stamp
        0.00 version
           1 ordinal base
           1 number of functions
           1 number of names

    ordinal hint RVA      name
```

(continues on next page)

(continued from previous page)

```
          1    0 00001000 GetGreeting

  Summary

        1000 .rdata
        1000 .reloc
        1000 .text

D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
```

To reiterate, this command lists the functions that other dlls can import into their program for use using *LoadLibrary*

DLL Depencencies

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>␣
↪dumpbin -dependents PrintGreeting.exe
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file PrintGreeting.exe

File Type: EXECUTABLE IMAGE

  Image has the following dependencies:

    KERNEL32.dll

  Summary

        2000 .data
        6000 .rdata
        1000 .reloc
        D000 .text
```

DLL Imports

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin -imports PrintGreeting.exe
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file PrintGreeting.exe

File Type: EXECUTABLE IMAGE

  Section contains the following imports:

    KERNEL32.dll
                40E000 Import Address Table
                4133AC Import Name Table
                     0 time date stamp
```

(continues on next page)

(continued from previous page)

```
                      0 Index of first forwarder reference

                    1AB FreeLibrary
                    2AE GetProcAddress
                    3C3 LoadLibraryExW
                    44D QueryPerformanceCounter
                    218 GetCurrentProcessId
                    21C GetCurrentThreadId
                    2E9 GetSystemTimeAsFileTime
                    363 InitializeSListHead
                    37F IsDebuggerPresent
                    5AD UnhandledExceptionFilter
                    56D SetUnhandledExceptionFilter
                    2D0 GetStartupInfoW
                    386 IsProcessorFeaturePresent
                    278 GetModuleHandleW
                    217 GetCurrentProcess
                    58C TerminateProcess
                    611 WriteConsoleW
                    4D3 RtlUnwind
                    261 GetLastError
                    532 SetLastError
                    131 EnterCriticalSection
                    3BD LeaveCriticalSection
                    110 DeleteCriticalSection
                    35F InitializeCriticalSectionAndSpinCount
                    59E TlsAlloc
                    5A0 TlsGetValue
                    5A1 TlsSetValue
                    59F TlsFree
                    462 RaiseException
                    2D2 GetStdHandle
                    612 WriteFile
                    274 GetModuleFileNameW
                    15E ExitProcess
                    277 GetModuleHandleExW
                    1D6 GetCommandLineA
                    1D7 GetCommandLineW
                    24E GetFileType
                    345 HeapAlloc
                    349 HeapFree
                    175 FindClose
                    17B FindFirstFileExW
                    18C FindNextFileW
                    38B IsValidCodePage
                    1B2 GetACP
                    297 GetOEMCP
                    1C1 GetCPInfo
                    3EF MultiByteToWideChar
                    5FE WideCharToMultiByte
                    237 GetEnvironmentStringsW
                    1AA FreeEnvironmentStringsW
                    514 SetEnvironmentVariableW
                    54A SetStdHandle
                    2D7 GetStringTypeW
                     9B CompareStringW
                    3B1 LCMapStringW
```

(continues on next page)

(continued from previous page)

```
                  2B4 GetProcessHeap
                  24C GetFileSizeEx
                  523 SetFilePointerEx
                  1EA GetConsoleCP
                  1FC GetConsoleMode
                  34E HeapSize
                  34C HeapReAlloc
                  19F FlushFileBuffers
                   86 CloseHandle
                   CB CreateFileW
                  109 DecodePointer

  Summary

        2000 .data
        6000 .rdata
        1000 .reloc
        D000 .text
```

6.2.2 Implicit Linking

Before, we use explicit linking to LoadLibrary and GetProcAddress for specific functions from the library we were using. Now we look at implicit linking.

Where explicit linking means you physically load the library in your program, with implicit linking you are providing a *.lib file, which contains the information needed for a program to implicitely link. Remember that this .lib is not the same as that produced when building a static library. Instead, it is a stub file that gets used to create function pointers automatically.

We want this to work:

```cpp
// PrintGreetingImplicityLinking.cpp
#include <stdio.h>

extern "C" const char* __cdecl GetGreeting();

int main(){
    puts(GetGreeting());
}
```

You can use

```
dumpbin -all Hello.lib
```

To look in detail at the *lib file. It gives us information such as which functions are available for linking, where they live etc.

We can compile and link:

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>cl -
↪c PrintGreetingImplicityLinking.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.26.28806 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

PrintGreetingImplicityLinking.cpp
```

(continues on next page)

(continued from previous page)

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪link PrintGreetingImplicityLinking.obj Hello.lib
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪PrintGreetingImplicityLinking.exe
Hello, C++ Programmers!
```

We can look at its dependents:

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /dependents PrintGreetingImplicityLinking.exe
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file PrintGreetingImplicityLinking.exe

File Type: EXECUTABLE IMAGE

  Image has the following dependencies:

    Hello.dll
    KERNEL32.dll

  Summary

        2000 .data
        6000 .rdata
        1000 .reloc
        D000 .text
```

Relealing that our PrintGreetingImplicitlLinking.exe depends on both Hello.dll and KERNEL32.dll, where our explicitely linked program only depended on KERNEL32.dll.

We can check our imports:

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /imports PrintGreetingImplicityLinking.exe
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file PrintGreetingImplicityLinking.exe

File Type: EXECUTABLE IMAGE

  Section contains the following imports:

    Hello.dll
                40E000 Import Address Table
                4133A0 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

```

(continues on next page)

(continued from previous page)

```
                   0 GetGreeting

    KERNEL32.dll
                40E008 Import Address Table
                4133A8 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

                  44D QueryPerformanceCounter
                  218 GetCurrentProcessId
                  21C GetCurrentThreadId
                  2E9 GetSystemTimeAsFileTime
                  363 InitializeSListHead
                  37F IsDebuggerPresent
                  5AD UnhandledExceptionFilter
                  56D SetUnhandledExceptionFilter
                  2D0 GetStartupInfoW
                  386 IsProcessorFeaturePresent
                  278 GetModuleHandleW
                  217 GetCurrentProcess
                  58C TerminateProcess
                  611 WriteConsoleW
                  4D3 RtlUnwind
                  261 GetLastError
                  532 SetLastError
                  131 EnterCriticalSection
                  3BD LeaveCriticalSection
                  110 DeleteCriticalSection
                  35F InitializeCriticalSectionAndSpinCount
                  59E TlsAlloc
                  5A0 TlsGetValue
                  5A1 TlsSetValue
                  59F TlsFree
                  1AB FreeLibrary
                  2AE GetProcAddress
                  3C3 LoadLibraryExW
                  462 RaiseException
                  2D2 GetStdHandle
                  612 WriteFile
                  274 GetModuleFileNameW
                  15E ExitProcess
                  277 GetModuleHandleExW
                  1D6 GetCommandLineA
                  1D7 GetCommandLineW
                  24E GetFileType
                  345 HeapAlloc
                  349 HeapFree
                  175 FindClose
                  17B FindFirstFileExW
                  18C FindNextFileW
                  38B IsValidCodePage
                  1B2 GetACP
                  297 GetOEMCP
                  1C1 GetCPInfo
                  3EF MultiByteToWideChar
                  5FE WideCharToMultiByte
                  237 GetEnvironmentStringsW
```

(continues on next page)

(continued from previous page)

```
                  1AA FreeEnvironmentStringsW
                  514 SetEnvironmentVariableW
                  54A SetStdHandle
                  2D7 GetStringTypeW
                   9B CompareStringW
                  3B1 LCMapStringW
                  2B4 GetProcessHeap
                  24C GetFileSizeEx
                  523 SetFilePointerEx
                  1EA GetConsoleCP
                  1FC GetConsoleMode
                  34E HeapSize
                  34C HeapReAlloc
                  19F FlushFileBuffers
                   86 CloseHandle
                   CB CreateFileW
                  109 DecodePointer

  Summary

        2000 .data
        6000 .rdata
        1000 .reloc
        D000 .text
```

Which indicates that we import our GetGreeting function from Hello.lib/Hello.dll.

6.2.3 Exporting from a DLL

We create a new example to work with.

```
// Numbers.cpp
extern "C" int GetOne() {return 1;}
extern "C" int GetTwo() {return 2;}
extern "C" int GetThree() {return 3;}
```

Lets compile:

```
cl -c Numbers.cpp
```

We have 4 options for exporting these function to make them available for

Export flag command line

So far we've been using Export.

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪link Numbers.obj /NOENTRY /DLL /EXPORT:GetOne /EXPORT:GetTwo /EXPORT:GetThree
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

   Creating library Numbers.lib and object Numbers.exp

D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /exports Numbers.dll
```

(continues on next page)

(continued from previous page)

```
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file Numbers.dll

File Type: DLL

  Section contains the following exports for Numbers.dll

    00000000 characteristics
    FFFFFFFF time date stamp
        0.00 version
           1 ordinal base
           3 number of functions
           3 number of names

    ordinal hint RVA      name

          1    0 00001000 GetOne
          2    1 00001020 GetThree
          3    2 00001010 GetTwo

  Summary

        1000 .rdata
        1000 .text
```

We can also export under alias's.

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪link Numbers.obj /NOENTRY /DLL /EXPORT:GetOne /EXPORT:GetTwo /EXPORT:GetThree /
↪EXPORT:GetOnePlusTwo=GetThree
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

   Creating library Numbers.lib and object Numbers.exp

D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /exports Numbers.dll
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file Numbers.dll

File Type: DLL

  Section contains the following exports for Numbers.dll

    00000000 characteristics
    FFFFFFFF time date stamp
        0.00 version
           1 ordinal base
           4 number of functions
           4 number of names
```

(continues on next page)

(continued from previous page)

```
    ordinal hint RVA      name

          1    0 00001000 GetOne
          2    1 00001020 GetOnePlusTwo
          3    2 00001020 GetThree
          4    3 00001010 GetTwo

  Summary

        1000 .rdata
        1000 .text
```

Note: GetOnePlusTwo and GetThree are the same function with a different name. They are at the same memory address.

Using a def file

In a new file, Numbers.def, put the following:

```
LIBRARY Numbers
EXPORTS
        GetOne
        GetTwo PRIVATE
        GetOnePlusTwo=GetThree
```

Now we can link with :

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪link Numbers.obj /DLL /NOENTRY /DEF:Numbers.def
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

   Creating library Numbers.lib and object Numbers.exp

D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /exports Numbers.lib
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file Numbers.lib

File Type: LIBRARY

     Exports

       ordinal    name

                  _GetOne
                  _GetOnePlusTwo

  Summary
```

(continues on next page)

(continued from previous page)

```
          C3 .debug$S
          14 .idata$2
          14 .idata$3
           4 .idata$4
           4 .idata$5
           C .idata$6
```

Inside your code

Another option is to declare exports inside your code. Take a look at Numbers2.cpp.

```
extern "C" __declspec(dllexport) int GetOne() { return 1;}
extern "C" __declspec(dllexport) int GetTwo() { return 2;}
extern "C" __declspec(dllexport) int GetThree() { return 3;}
```

We use __declspec(export) to do that same as what we were previously doing on the command line. The

```
D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>cl -
↪c Numbers2.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.26.28806 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

Numbers2.cpp

D:\TestStaticIntoSharedLinking\cmake-build-release-visual-studio\dynamic_lib\test>
↪dumpbin /EXPORTS Numbers2.dll
Microsoft (R) COFF/PE Dumper Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file Numbers2.dll

File Type: DLL

  Section contains the following exports for Numbers2.dll

    00000000 characteristics
    FFFFFFFF time date stamp
        0.00 version
           1 ordinal base
           3 number of functions
           3 number of names

    ordinal hint RVA      name

          1    0 00001000 GetOne
          2    1 00001020 GetThree
          3    2 00001010 GetTwo

  Summary

        1000 .rdata
        1000 .text
```

Declspec export merely tells the compiler to pretend that it got the exports from the command line. They do the same job but its more convenient.

Pragma

Pragma directives can also be used to achieve the same, though this is not often used. So Numbers3.cpp looks like this.

6.2.4 What happens when we load a DLL?

There are 5 steps, basically:

1. Find the dll (Hello.dll)
2. Map Hello.dll into memory
3. Load any DLLs on which Hello.dll depends
4. Bind imports from DLLs on which Hello.dll depends
5. Call the entry point for Hello.dll to let it initialize itself.

Find the DLL

When we do

```
HMODULE HelloDll = LoadLibraryExW(L"Hello.dll", nullptr, o);
```

How does the loader know where to find *Hello.dll*?

If we passed an absolute path to *LoadLibraryExW*, this is easy as if its there it'll be loaded, if not it'll fail. Note, you can load the same library into the same script from two different drives (C Vs D), but not two libraries with the same name from the same drive.

If its not an absolute path then the first thing that happens is the loader will look to see whether the dll is a system dll. These are always loaded from the same place for security. These are well known to the OS and the same version of the library will always be loaded. For instance, kernel32.dll or ole32.dll. This mechanism prevents dll hijacking.

If the dll is not in this small list of libraries, the loader will continue with the search process. This is the search process:

1. The directory from which the application is loaded
2. The system directoy (C:WindowsSystem32or C:WindowsSysWOW64)
3. The 16-bit system directory (C:WindowsSystem)
4. The Windows Directory (C:Windows)
5. The current directory
6. The directories listed in %PATH% environment variable.

Once found, the search stops.

This process is highly customizable. For instance:

1. DLL Redirection (.local)
2. Side-by-size components
3. add to %PATH%
4. AddDllDirectory
5. LoadLibraryEx Flags

Do some googling on these.

Map the DLL into Memory

The loader needs to

1. Open the DLL file and read the image size
2. Allocate a contiguous, page aligned block of memory of that size
3. Copy the contents of each section into the appropriate area of that block of memory

Relocation

DLLs have a preferred base address. If the dll does not get loaded into its preferred base address then the pointers in the dll will be pointing to random slots of memory. Relocation fixes this.

Load Dependencies and Bind Imports

For each DLL dependency:

1. load the DLL
2. Get the required imports to fill out the function pointer tables.

Initialize the DLL

DLLs have an optional entry point where it can do some initialization. Conventially this is called *DllMain* but can be called anything.

Here is the signature.

```
BOOL WINAPI DllMain(HINSTANCE instance, DWORD reason, LPVOID reserved);
```

Where:

- instance = the DLL handle returned from LoadLibrary
- **reason = indication of why the loaded is calling the entry point**
  - DLL_PROCESS_ATTACH = Called once, when DLL is loaded
  - DLL_PROCESS_DETACH = Called once, when DLL is unloaded
  - DLL_THREAD_ATTACH = Called each time a thread starts running
  - DLL_THREAD_DETACH = Called each time a thread stops running
- reserve = more information for process attach or detach.

Returns True or False depending on load success.

Calls to DllMain are syncronized by a gloval lock called the Loader Lock. So only 1 thread can be initializing a dll at one time.

Debugging DLL Load Failures

What if Hello.dll did not exist? Then you would get an error. How do you debug this?

One way is to use a program called gflags.

Here I deleted Hello.dll. Now when we run a program that uses Hello.dll we get and error.

Importing

We've already seen *__declspec(dllexport)* which is used inside our source files to allow other programs access to the public interface. *__declspec(dllimport)* also exists, and this is used inside programs that *use* a dll.

For instance, see *NumbersCaller.cpp*.

The *__declspec(dllimport)* statement tells the compiler than this function is going to be imported. This is more efficient because the compiler can do things a little differently.

Exporting Data

You can export variables as well as functions. When you do this you need to use __declspec(dllimport).

Exporting C++ classes

This is possible. When you use *__declspec(dllexport)* on a class, rather than a function, all the members of the class get exported.

However, You are NOT recommended to do exports on classes. You are too dependent on a compiler. This will be hard to debug and will probably do wrong.

6.3 Powershell

Open windows explorer from this directory. ii is short for Invoke-Item

```
ii .
```

CHAPTER 7

Travis

Some useful links:

referece: https://config.travis-ci.com/ syntax schema: http://json.schemastore.org/travis some commands: https://devhints.io/travis https://github.com/travis-ci/docs-travis-ci-com/issues/2004

CHAPTER 8

Random Notes

CHAPTER 9

Indices and tables

- genindex
- modindex
- search