Everything You Ever Wanted to Know about DLLs

JAMES MCNELLIS
SENIOR SOFTWARE ENGINEER
MICROSOFT / WINDOWS DEBUGGERS
@JAMESMCNELLIS

a few things ~~Everything~~ You probably didn't ~~Ever Wanted~~ to Know about DLLs

JAMES MCNELLIS
SENIOR SOFTWARE ENGINEER
MICROSOFT / WINDOWS DEBUGGERS
@JAMESMCNELLIS

We are going to talk about…

- How to build a DLL
- How to use a DLL in a program
- What's inside of a DLL
- Explicit and implicit linking
- What happens when you load a DLL
- How to diagnose DLL load failures
- Various ways to specify what a DLL exports
- Data exports
- Delay loading
- C++ and DLLs
- Threads and TLS
- DLL Hell

What We're Going to Talk About

We are not going to talk about…

- Dynamic libraries or shared objects on other platforms
- .NET assemblies or other .NET topics
- Resource DLLs

We will mention many things that we wont discuss in depth…

- …but we'll include sufficient information if you want to learn more about them afterwards

What We're Not Going to Talk About

What and Why?

A DLL is a Dynamic Link Library

It's a library…

- …that contains code and data
- …that can be loaded dynamically at runtime
- …that can be shared or reused by multiple programs

Most "normal" DLLs have a .dll file extension.

What Is a DLL?

Multiple programs can share code and data without each program having its own copy

- Can reduce disk space usage
- Can reduce memory usage

Can defer decision of whether to load functionality until runtime

- Perhaps you may not always need some functionality
- Perhaps you want to support open-ended extensibility (e.g. plugins)

Maintainability benefits

- Componentization via DLLs
- Improved serviceability (bug fixes, security patching, etc.)
- Improved maintainability

Why Use DLLs?

More complicated software distribution

- If you build everything into a single EXE, it's pretty easy to install

Increased potential for incompatibilities (DLL Hell)

Impossible to optimize code across DLL boundaries

- Every call across a DLL boundary is necessarily an indirect call
- (But there are performance benefits to using DLLs too)

Why Not Use DLLs?

Let’s Build a Little DLL…

A:\>

Hello.dll

```
A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>
```

Hello.dll

```
A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>cl /c Hello.cpp
Hello.cpp

A:\>
```

On most future slides, we'll skip the compilation step for brevity

Hello.dll

```
A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>cl /c Hello.cpp
Hello.cpp

A:\>link Hello.obj
         /DLL
         /NOENTRY
         /EXPORT:GetGreeting
Creating library Hello.lib...

A:\>
```

Hello.dll

```
A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>cl /c Hello.cpp
Hello.cpp

A:\>link Hello.obj
         /DLL
         /NOENTRY
         /EXPORT:GetGreeting
Creating library Hello.lib...

A:\>
```

Hello.dll

Hello.dll

A:\>

Hello.dll

```
A:\>Hello.dll
The system cannot execute the specified program.

A:\>
```

Hello.dll

```
A:\>
```

PrintGreeting.exe

```
A:\>type PrintGreeting.cpp
#include <stdio.h>
#include <Windows.h>

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

    // char const* __cdecl GetGreeting();
    using GetGreetingType = char const* (__cdecl*)();

    GetGreetingType const GetGreeting =
        reinterpret_cast<GetGreetingType>(
            GetProcAddress(HelloDll, "GetGreeting"));

    puts(GetGreeting());

    FreeLibrary(HelloDll);
}
```

PrintGreeting.exe

```
A:\>
```

PrintGreeting.exe

```
A:\>cl PrintGreeting.cpp
PrintGreeting.cpp

A:\>
```

PrintGreeting.exe

```
A:\>cl PrintGreeting.cpp
PrintGreeting.cpp

A:\>PrintGreeting.exe
Hello, C++ Programmers!

A:\>
```

PrintGreeting.exe

```
A:\>cl PrintGreeting.cpp
PrintGreeting.cpp

A:\>PrintGreeting.exe
Hello, C++ Programmers!

A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>
```

PrintGreeting.exe

```
A:\>cl PrintGreeting.cpp
PrintGreeting.cpp

A:\>PrintGreeting.exe
Hello, C++ Programmers!

A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>
```

Let’s Take a Look Inside Hello.dll

A:\>

What’s Inside Hello.dll?

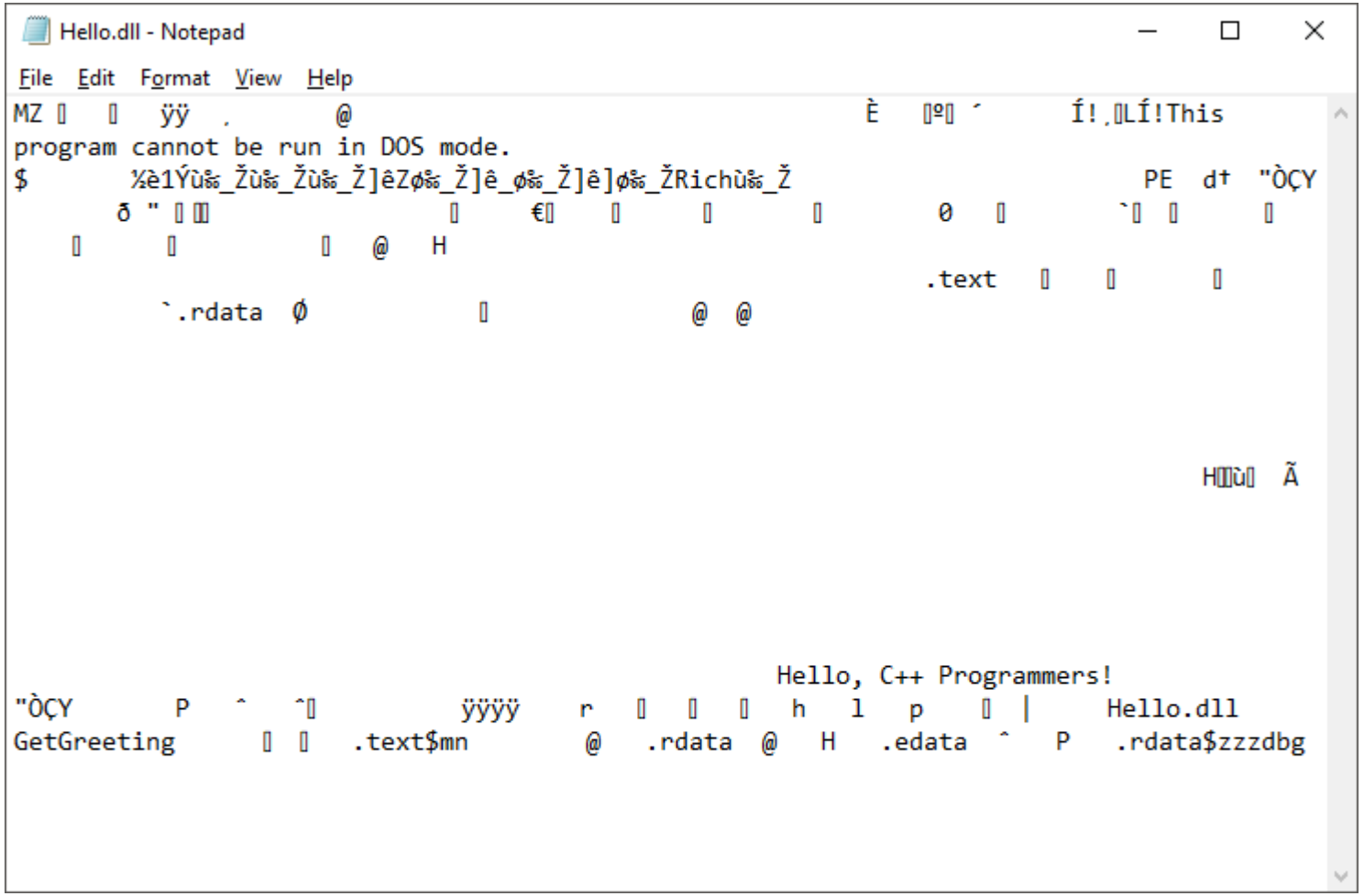
```
A:\>type Hello.dll
MZÉ                 ¶       @                                       ╚   ╢ ┤  =!╕L=!This program cannot
be run in DOS mode.
$ ¶       ╝Φ1▌·ë_Ä·ë_Ä·ë_Ä]ΩZÅ°ë_Ä]Ω_Å°ë_Ä]Ω]Å°ë_ÄRich·ë_Ä                          PE  då 1µ╝Y
≡ ¶

.text                   Ç                   `.rdata 0 ╧                 `                                   @   H
Hi·  ├                                                                  @  @
P   ê │  ê                        r                   h   l   p       │      Hello.dll GetGreeting
.text$mn        @   .rdata  @   H   .edata  ê   P   .rdata$zzzdbg

A:\>
```

What’s Inside Hello.dll?

A:\>

What’s Inside Hello.dll?

A:\>notepad Hello.dll



What’s Inside Hello.dll?

DOS Stub	• Valid DOS program (not very useful anymore…)
PE Signtaure	• PE\0\0
COFF File Header	• Common file header
“Optional” Header	• Image-specific file headers
Section Headers	• Contain information about the “sections” in the DLL
Sections 0..N	• Contains the actual code, data, and resources in the DLL

DLL File Structure

DOS Stub	• Valid DOS program (not very useful anymore…)
PE Signtaure	• PE\0\0
COFF File Header	• Common file header
“Optional” Header	• Image-specific file headers
Section Headers	• Contain information about the “sections” in the DLL
Sections 0..N	• Contains the actual code, data, and resources in the DLL

DLL File Structure

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 C8 00 00 00  ............È...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  BD E8 31 DD F9 89 5F 8E F9 89 5F 8E F9 89 5F 8E  ½è1Ýù‰_Žù‰_Žù‰_Ž
00000090  5D EA 5A 8F F8 89 5F 8E 5D EA 5F 8F F8 89 5F 8E  ]êZ.ø‰_Ž]ê_.ø‰_Ž
000000A0  5D EA 5D 8F F8 89 5F 8E 52 69 63 68 F9 89 5F 8E  ]ê].ø‰_ŽRichù‰_Ž
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 50 45 00 00 [continued]  ........PE..
```

The DOS Stub at the beginning of the file…

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 C8 00 00 00  ............È...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  BD E8 31 DD F9 89 5F 8E F9 89 5F 8E F9 89 5F 8E  ½è1Ýù‰_Žù‰_Žù‰_Ž
00000090  5D EA 5A 8F F8 89 5F 8E 5D EA 5F 8F F8 89 5F 8E  ]êZ.ø‰_Ž]ê_.ø‰_Ž
000000A0  5D EA 5D 8F F8 89 5F 8E 52 69 63 68 F9 89 5F 8E  ]ê].ø‰_ŽRichù‰_Ž
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 50 45 00 00 [continued]  ........PE..
```

…and the PE Signature

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 C8 00 00 00  ............È...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  BD E8 31 DD F9 89 5F 8E F9 89 5F 8E F9 89 5F 8E  ½è1Ýù‰_Žù‰_Žù‰_Ž
00000090  5D EA 5A 8F F8 89 5F 8E 5D EA 5F 8F F8 89 5F 8E  ]êZ.ø‰_Ž]ê_.ø‰_Ž
000000A0  5D EA 5D 8F F8 89 5F 8E 52 69 63 68 F9 89 5F 8E  ]ê].ø‰_ŽRichù‰_Ž
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 50 45 00 00 [continued]  ........PE..
```

How do we find the PE Signature?

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 C8 00 00 00  ............È...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  BD E8 31 DD F9 89 5F 8E F9 89 5F 8E F9 89 5F 8E  ½è1Ýù‰_Žù‰_Žù‰_Ž
00000090  5D EA 5A 8F F8 89 5F 8E 5D EA 5F 8F F8 89 5F 8E  ]êZ.ø‰_Ž]ê_.ø‰_Ž
000000A0  5D EA 5D 8F F8 89 5F 8E 52 69 63 68 F9 89 5F 8E  ]ê].ø‰_ŽRichù‰_Ž
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 50 45 00 00 [continued]  ........PE..
```

Look at offset 3C…

…that tells us the offset (C8) of the PE Signature

How do we find the PE Signature?

DOS Stub	• Valid DOS program (not very useful anymore…)
PE Signtaure	• PE\0\0
COFF File Header	• Common file header
“Optional” Header	• Image-specific file headers
Section Headers	• Contain information about the “sections” in the DLL
Sections 0..N	• Contains the actual code, data, and resources in the DLL

DLL File Structure

The COFF File Header

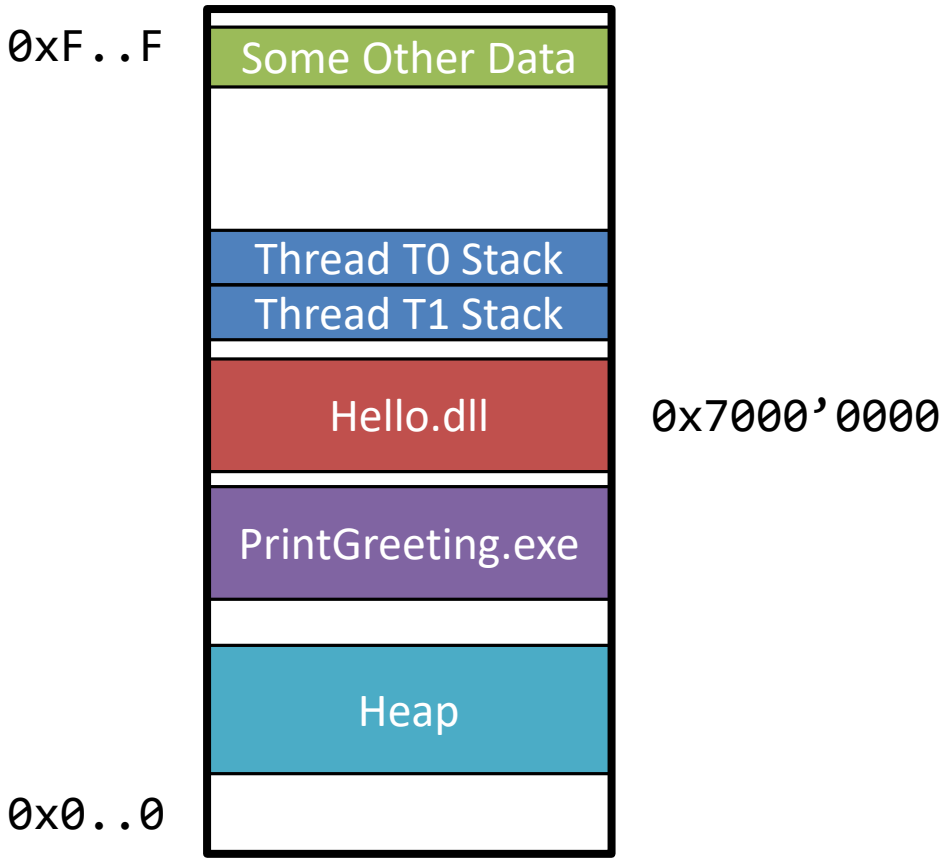
```
A:\>dumpbin /headers Hello.dll
Dump of file Hello.dll

PE signature found
File Type: DLL

FILE HEADER VALUES
        8664 machine (x64)
           2 number of sections
    59BDE631 time date stamp Sat Sep 16 20:04:17 2017
          F0 size of optional header
        2022 characteristics
               Executable
               Application can handle large (>2GB) addresses
               DLL
```

The COFF File Header

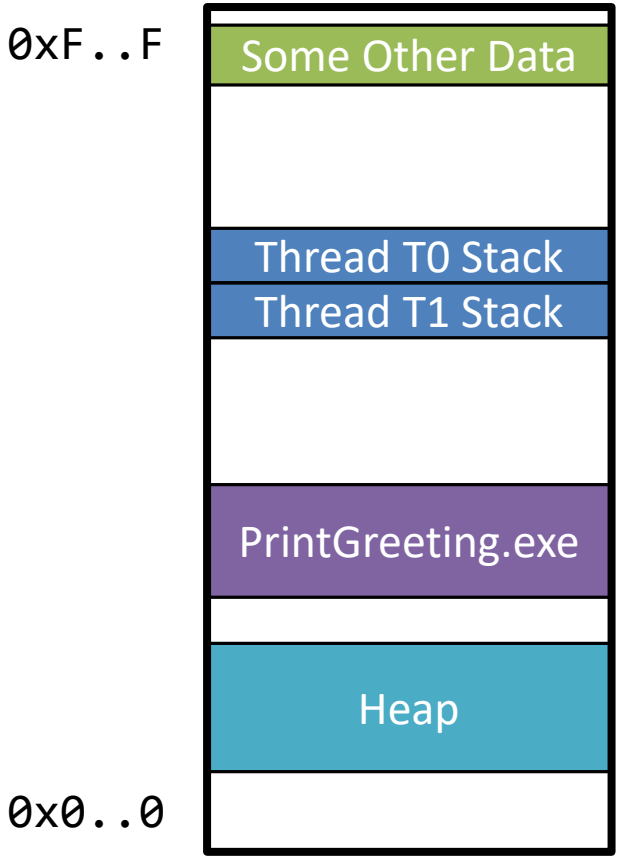
```
OPTIONAL HEADER VALUES
         20B magic # (PE32+)
           0 entry point
    70000000 image base (70000000 to 70002FFF)
        1000 section alignment
         200 file alignment
        3000 size of image
         400 size of headers
         160 DLL characteristics
               High Entropy Virtual Addresses
               Dynamic base
               NX compatible
          10 number of directories
        2040 [      48] RVA [size] of Export Directory
           0 [       0] RVA [size] of Import Directory
           0 [       0] RVA [size] of Resource Directory
           0 [       0] RVA [size] of Exception Directory
        2020 [      1C] RVA [size] of Debug Directory
        [...and more empty directories...]
```

The “Optional” Header

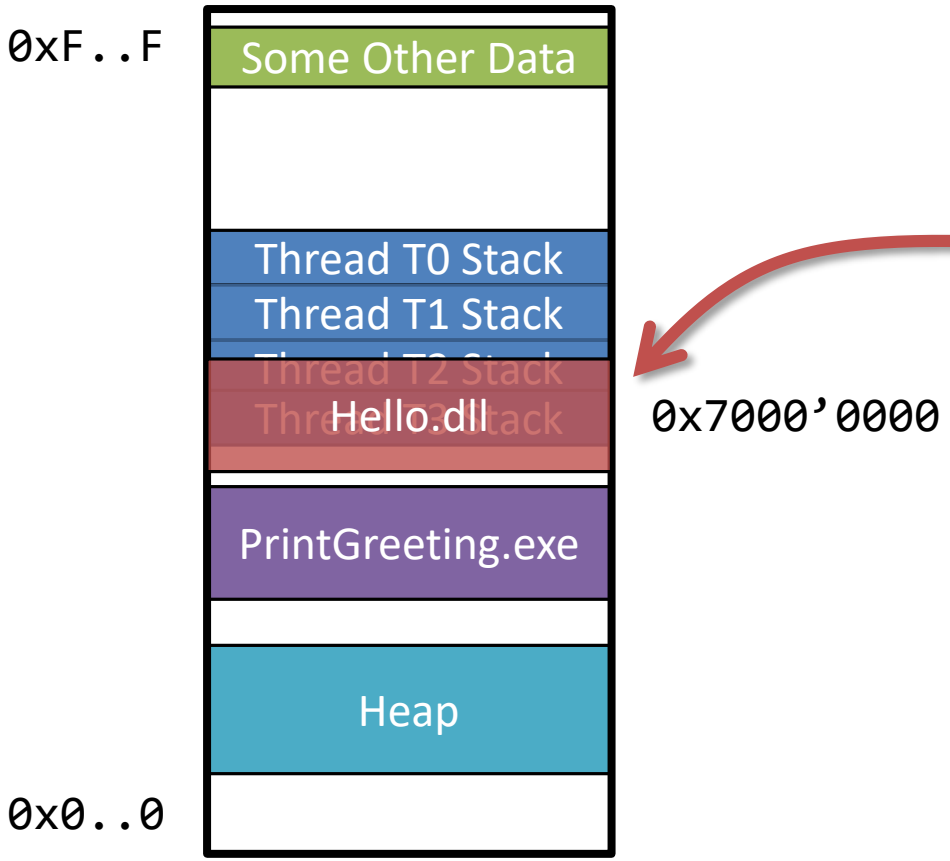
Addressing Within a DLL



Addressing Within a DLL

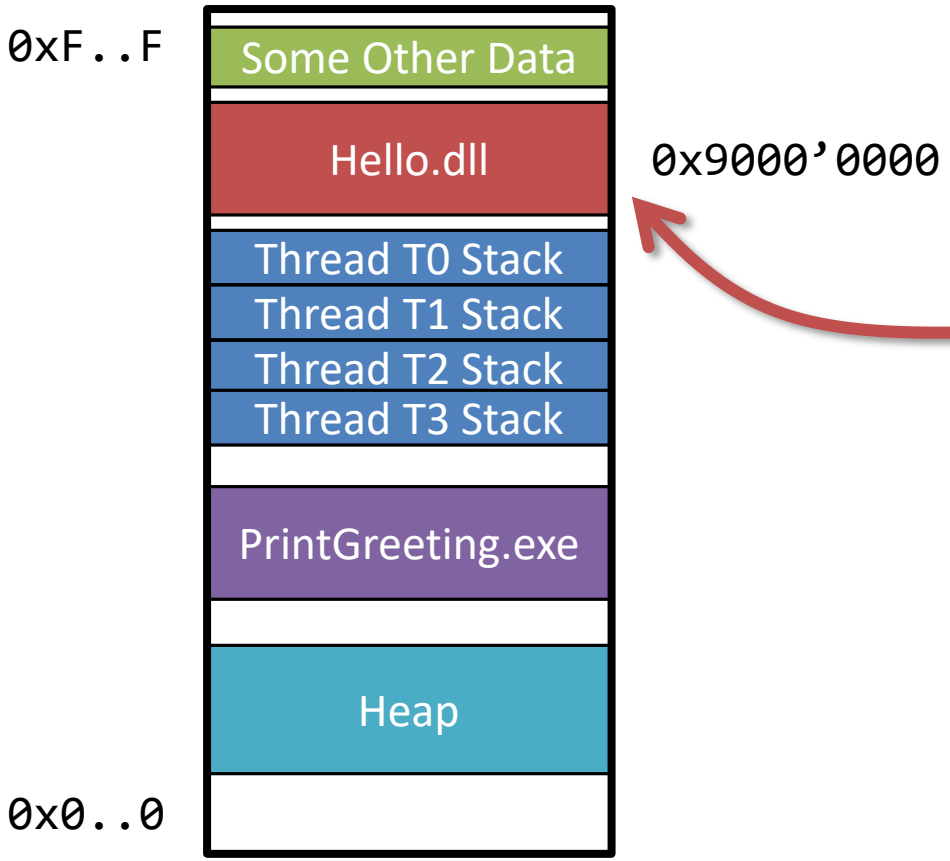


Addressing Within a DLL



We can't load at that address…
…there's already something there.

Addressing Within a DLL



So perhaps the DLL will get loaded here instead.

Addressing Within a DLL

We need a way to address things within a DLL…
…without relying on the actual address at which it is loaded

Relative Virtual Address (RVA):
An *offset* from the beginning of the DLL

Address in Memory = DLL Base Address + RVA

RVA = Address in Memory – DLL Base Address

For example, if…

- function GetGreeting() in Hello.dll has RVA = 0x2000, and
- Hello.dll gets loaded at address 0x7000’0000,

…then function GetGreeting() will be at address 0x7000’2000.
(0x7000’0000 + 0x2000 = 0x7000’2000)

Relative Virtual Addresses (RVAs)

```
OPTIONAL HEADER VALUES
         20B magic # (PE32+)
           0 entry point
    70000000 image base (70000000 to 70002FFF)
        1000 section alignment
         200 file alignment
        3000 size of image
         400 size of headers
         160 DLL characteristics
               High Entropy Virtual Addresses
               Dynamic base
               NX compatible
          10 number of directories
        2040 [      48] RVA [size] of Export Directory
           0 [       0] RVA [size] of Import Directory
           0 [       0] RVA [size] of Resource Directory
           0 [       0] RVA [size] of Exception Directory
        2020 [      1C] RVA [size] of Debug Directory
        [...and more empty directories...]
```

The “Optional” Header

```
SECTION HEADER #1
   .text name
       8 virtual size
    1000 virtual address (70001000 to 70001007)
     200 size of raw data
     400 file pointer to raw data (00000400 to 000005FF)
60000020 flags
         Code
         Execute Read
```

Section Header #1: .text Section

```
SECTION HEADER #2
  .rdata name
       D8 virtual size
     2000 virtual address (70002000 to 700020D7)
      200 size of raw data
      600 file pointer to raw data (00000600 to 000007FF)
 40000040 flags
          Initialized Data
          Read Only
```

```
Optional Header Directories
2040 [      48] RVA [size] of Export Directory
2020 [      1C] RVA [size] of Debug Directory
```

Section Header #2: .rdata Section

The DLL will occupy 0x3000 bytes (3 pages) in memory when loaded into a process…

- one page containing the headers
- one page containing the .text section
- one page containing the .rdata section

(These aren't the only possible sections: There are other kinds of sections we might find in a DLL)

The DLL has additional metadata in a pair of directories…

- a debug directory
- an export directory

…and this data is contained in the .rdata section.

So, from Hello.dll's headers we know…

DOS Stub	• Valid DOS program (not very useful anymore…)
PE Signtaure	• PE\0\0
COFF File Header	• Common file header
“Optional” Header	• Image-specific file headers
Section Headers	• Contain information about the “sections” in the DLL
Sections 0..N	• Contains the actual code, data, and resources in the DLL

DLL File Structure

Section #1: .text

```
A:\>dumpbin /rawdata /section:.text Hello.dll
  70001000: 48 8D 05 F9 0F 00 00 C3                                H..ù...Ã

A:\>
```

Section #1: .text

```
A:\>dumpbin /rawdata /section:.text Hello.dll
  70001000: 48 8D 05 F9 0F 00 00 C3                          H..ù...Ã

A:\>dumpbin /disasm /section:.text Hello.dll
  70001000: 48 8D 05 F9 0F 00 00  lea rax,[70002000h]
  70001007: C3                    ret

A:\>
```

Section #1: .text

Section #2: .rdata

```
A:\>dumpbin /rawdata /section:.rdata Hello.dll
  70002000: 48 65 6C 6C 6F 2C 20 43 2B 2B 20 50 72 6F 67 72  Hello, C++ Progr
  70002010: 61 6D 6D 65 72 73 21 00 00 00 00 00 00 00 00 00  ammers!.........
  70002020: 00 00 00 00 31 E6 BD 59 00 00 00 00 0D 00 00 00  ....1æ½Y........
  70002030: 50 00 00 00 88 20 00 00 88 06 00 00 00 00 00 00  P.... ..........
  70002040: 00 00 00 00 FF FF FF FF 00 00 00 00 72 20 00 00  ....ÿÿÿÿ....r ..
  70002050: 01 00 00 00 01 00 00 00 01 00 00 00 68 20 00 00  ............h ..
  70002060: 6C 20 00 00 70 20 00 00 00 10 00 00 7C 20 00 00  l ..p ......| ..
  70002070: 00 00 48 65 6C 6C 6F 2E 64 6C 6C 00 47 65 74 47  ..Hello.dll.GetG
  70002080: 72 65 65 74 69 6E 67 00 00 00 00 00 00 10 00 00  reeting.........
  70002090: 08 00 00 00 2E 74 65 78 74 24 6D 6E 00 00 00 00  .....text$mn....
  700020A0: 00 20 00 00 40 00 00 00 2E 72 64 61 74 61 00 00  . ..@....rdata..
  700020B0: 40 20 00 00 48 00 00 00 2E 65 64 61 74 61 00 00  @ ..H....edata..
  700020C0: 88 20 00 00 50 00 00 00 2E 72 64 61 74 61 24 7A  . ..P....rdata$z
  700020D0: 7A 7A 64 62 67 00 00 00                          zzdbg...

A:\>
```

Section #2: .rdata

```
A:\>dumpbin /rawdata /section:.rdata Hello.dll
  70002000: 48 65 6C 6C 6F 2C 20 43 2B 2B 20 50 72 6F 67 72  Hello, C++ Progr
  70002010: 61 6D 6D 65 72 73 21 00 00 00 00 00 00 00 00 00  ammers!.........
  70002020: 00 00 00 00 31 E6 BD 59 00 00 00 00 0D 00 00 00  ....1æ½Y........
  70002030: 50 00 00 00 88 20 00 00 88 06 00 00 00 00 00 00  P.... ..........
  70002040: 00 00 00 00 FF FF FF FF 00 00 00 00 72 20 00 00  ....ÿÿÿÿ....r ..
  70002050: 01 00 00 00 01 00 00 00 01 00 00 00 68 20 00 00  ............h ..
  70002060: 6C 20 00 00 70 20 00 00 00 10 00 00 7C 20 00 00  l ..p ......| ..
  70002070: 00 00 48 65 6C 6C 6F 2E 64 6C 6C 00 47 65 74 47  ..Hello.dll.GetG
  70002080: 72 65 65 74 69 6E 67 00 00 00 00 00 00 10 00 00  reeting.........
  70002090: 08 00 00 00 2E 74 65 78 74 24 6D 6E 00 00 00 00  .....text$mn....
  700020A0: 00 20 00 00 40 00 00 00 2E 72 64 61 74 61 00 00  . ..@....rdata..
  700020B0: 40 20 00 00 48 00 00 00 2E 65 64 61 74 61 00 00  @ ..H....edata..
  700020C0: 88 20 00 00 50 00 00 00 2E 72 64 61 74 61 24 7A  . ..P....rdata$z
  700020D0: 7A 7A 64 62 67 00 00 00                          zzdbg...
```

```
Optional Header Directories
2040 [      48] RVA [size] of Export Directory
2020 [      1C] RVA [size] of Debug Directory
```

Section #2: .rdata

```
A:\>dumpbin /rawdata /section:.rdata Hello.dll
  70002000: 48 65 6C 6C 6F 2C 20 43 2B 2B 20 50 72 6F 67 72  Hello, C++ Progr
  70002010: 61 6D 6D 65 72 73 21 00 00 00 00 00 00 00 00 00  ammers!.........
  70002020: 00 00 00 00 31 E6 BD 59 00 00 00 00 0D 00 00 00  ....1æ½Y........
  70002030: 50 00 00 00 88 20 00 00 88 06 00 00 00 00 00 00  P.... ..........
  70002040: 00 00 00 00 FF FF FF FF 00 00 00 00 72 20 00 00  ....ÿÿÿÿ....r ..
  70002050: 01 00 00 00 01 00 00 00 01 00 00 00 68 20 00 00  ............h ..
  70002060: 6C 20 00 00 70 20 00 00 00 10 00 00 7C 20 00 00  l ..p ......| ..
  70002070: 00 00 48 65 6C 6C 6F 2E 64 6C 6C 00 47 65 74 47  ..Hello.dll.GetG
  70002080: 72 65 65 74 69 6E 67 00 00 00 00 00 00 10 00 00  reeting.........
  70002090: 08 00 00 00 2E 74 65 78 74 24 6D 6E 00 00 00 00  .....text$mn....
  700020A0: 00 20 00 00 40 00 00 00 2E 72 64 61 74 61 00 00  . ..@....rdata..
  700020B0: 40 20 00 00 48 00 00 00 2E 65 64 61 74 61 00 00  @ ..H....edata..
  700020C0: 88 20 00 00 50 00 00 00 2E 72 64 61 74 61 24 7A  . ..P....rdata$z
  700020D0: 7A 7A 64 62 67 00 00 00                          zzdbg...
```

```
Optional Header Directories
2040 [      48] RVA [size] of Export Directory
2020 [      1C] RVA [size] of Debug Directory
```

Section #2: .rdata

```
A:\>dumpbin /rawdata /section:.rdata Hello.dll
  70002000: 48 65 6C 6C 6F 2C 20 43 2B 2B 20 50 72 6F 67 72  Hello, C++ Progr
  70002010: 61 6D 6D 65 72 73 21 00 00 00 00 00 00 00 00 00  ammers!.........
  70002020: 00 00 00 00 31 E6 BD 59 00 00 00 00 0D 00 00 00  ....1æ½Y........
  70002030: 50 00 00 00 88 20 00 00 88 06 00 00 00 00 00 00  P.... ..........
  70002040: 00 00 00 00 FF FF FF FF 00 00 00 00 72 20 00 00  ....ÿÿÿÿ....r ..
  70002050: 01 00 00 00 01 00 00 00 01 00 00 00 68 20 00 00  ............h ..
  70002060: 6C 20 00 00 70 20 00 00 00 10 00 00 7C 20 00 00  l ..p ......| ..
  70002070: 00 00 48 65 6C 6C 6F 2E 64 6C 6C 00 47 65 74 47  ..Hello.dll.GetG
  70002080: 72 65 65 74 69 6E 67 00 00 00 00 00 00 10 00 00  reeting.........
  70002090: 08 00 00 00 2E 74 65 78 74 24 6D 6E 00 00 00 00  .....text$mn....
  700020A0: 00 20 00 00 40 00 00 00 2E 72 64 61 74 61 00 00  . ..@....rdata..
  700020B0: 40 20 00 00 48 00 00 00 2E 65 64 61 74 61 00 00  @ ..H....edata..
  700020C0: 88 20 00 00 50 00 00 00 2E 72 64 61 74 61 24 7A  . ..P....rdata$z
  700020D0: 7A 7A 64 62 67 00 00 00                          zzdbg...
```

```
Optional Header Directories
2040 [      48] RVA [size] of Export Directory
2020 [      1C] RVA [size] of Debug Directory
```

Section #2: .rdata

A:\>

Export Directory

```
A:\>dumpbin /exports Hello.dll
Section contains the following exports for Hello.dll

    ordinal hint RVA      name

          1    0 00001000 GetGreeting

A:\>
```

Export Directory

```
A:\>dumpbin /exports Hello.dll
Section contains the following exports for Hello.dll

    ordinal hint RVA      name

          1    0 00001000 GetGreeting

A:\>
```

GetGreeting() is located at RVA 1000

Putting It All Together

```
A:\>dumpbin /exports Hello.dll
Section contains the following exports for Hello.dll

    ordinal hint RVA      name

          1    0 00001000 GetGreeting

A:\>dumpbin /disasm /section:.text Hello.dll

  70001000: 48 8D 05 F9 0F 00 00  lea rax,[70002000h]
  70001007: C3                    ret

A:\>
```

GetGreeting() is located at RVA 1000

Return the address of whatever is at RVA 2000

Putting It All Together

```
A:\>dumpbin /exports Hello.dll
Section contains the following exports for Hello.dll

    ordinal hint RVA      name

          1    0 00001000 GetGreeting

A:\>dumpbin /disasm /section:.text Hello.dll

  70001000: 48 8D 05 F9 0F 00 00  lea rax,[70002000h]
  70001007: C3                    ret

A:\>dumpbin /rawdata /section:.rdata Hello.dll
RAW DATA #2
  70002000: 48 65 6C 6C 6F 2C 20 43 2B 2B 20 50 72 6F 67 72  Hello, C++ Progr
  70002010: 61 6D 6D 65 72 73 21 00                          ammers!.

A:\>
```

GetGreeting() is located at RVA 1000

Return the address of whatever is at RVA 2000

It's our greeting string!

Putting It All Together

Implicit Linking and Import Libraries

```
A:\>type PrintGreeting.cpp
#include <stdio.h>
#include <Windows.h>

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

    // char const* __cdecl GetGreeting();
    using GetGreetingType = char const* (__cdecl*)();

    GetGreetingType const GetGreeting =
        reinterpret_cast<GetGreetingType>(
            GetProcAddress(HelloDll, "GetGreeting"));

    puts(GetGreeting());

    FreeLibrary(HelloDll);
}
```

LoadLibraryExW is defined in a DLL…

GetProcAddress is also defined in a DLL…

Explicit Linking

Implicit Dependencies

```
A:\>dumpbin /dependents PrintGreeting.exe
Image has the following dependencies:

    KERNEL32.dll

A:\>
```

Implicit Dependencies

```
A:\>dumpbin /dependents PrintGreeting.exe
Image has the following dependencies:

    KERNEL32.dll

A:\>dumpbin /imports PrintGreeting.exe
Section contains the following imports:

    KERNEL32.dll
        1AD FreeLibrary
        2AE GetProcAddress
        3BB LoadLibraryExW
        [...and many more...]

A:\>
```

Implicit Dependencies

```
A:\>type PrintGreeting.cpp
#include <stdio.h>
#include <Windows.h>

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

    // char const* __cdecl GetGreeting();
    using GetGreetingType = char const* (__cdecl*)();

    GetGreetingType const GetGreeting =
        reinterpret_cast<GetGreetingType>(
            GetProcAddress(HelloDll, "GetGreeting"));

    puts(GetGreeting());

    FreeLibrary(HelloDll);
}
```

Explicit Linking

```
A:\>type PrintGreetingImplicit.cpp
#include <stdio.h>

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

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

A:\>
```

Implicit Linking

Import Libraries

```
A:\>link Hello.obj
         /DLL
         /NOENTRY
         /EXPORT:GetGreeting
Creating library Hello.lib...

A:\>
```

Hello.lib is an import library for Hello.dll

Import Libraries

```
A:\>link Hello.obj
         /DLL
         /NOENTRY
         /EXPORT:GetGreeting
Creating library Hello.lib...

A:\>dumpbin /exports Hello.lib
Dump of file Hello.lib

File Type: LIBRARY
    Exports
    ordinal    name
               GetGreeting

A:\>
```

Hello.lib is an import library for Hello.dll

Import Libraries

Import Libraries

```
A:\>dumpbin /all Hello.lib
public symbols
    636 GetGreeting
    636 __imp_GetGreeting

[...]

Archive member name at 636: Hello.dll/
    DLL name      : Hello.dll
    Symbol name   : GetGreeting
    Type          : code
    Hint          : 0

A:\>
```

```
// It's as if this library had...

using GGType = char const* (__cdecl*)();

GGType __imp_GetGreeting = /* magic */;

extern "C" char const* __cdecl GetGreeting()
{
    return __imp_GetGreeting();
}
```

Import Libraries

Implicit Linking

```
A:\>type PrintImplicit.cpp
#include <stdio.h>

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

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

A:\>
```

Implicit Linking

```
A:\>type PrintImplicit.cpp
#include <stdio.h>

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

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

A:\>link PrintImplicit.obj Hello.lib

A:\>
```

Implicit Linking

```
A:\>type PrintImplicit.cpp
#include <stdio.h>

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

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

A:\>link PrintImplicit.obj Hello.lib

A:\>PrintImplicit.exe
Hello, C++ Programmers!

A:\>
```

Implicit Linking

```
A:\>type PrintImplicit.cpp
#include <stdio.h>

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

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

A:\>link PrintImplicit.obj Hello.lib

A:\>PrintImplicit.exe
Hello, C++ Programmers!

A:\>
```

```
A:\>dumpbin /dependents PrintImplicit.exe
Image has the following dependencies:
    Hello.dll
    KERNEL32.dll

A:\>
```

Implicit Linking

```
A:\>type PrintImplicit.cpp
#include <stdio.h>

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

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

A:\>link PrintImplicit.obj Hello.lib

A:\>PrintImplicit.exe
Hello, C++ Programmers!

A:\>
```

```
A:\>dumpbin /dependents PrintImplicit.exe
Image has the following dependencies:
    Hello.dll
    KERNEL32.dll

A:\>dumpbin /imports PrintImplicit.exe
Section contains the following imports:
    Hello.dll
        0 GetGreeting

    KERNEL32.dll
    [...and many more...]

A:\>
```

Implicit Linking

Specifying Exports

Basic Exports

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>
```

Basic Exports

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /EXPORT:GetOne
         /EXPORT:GetTwo
         /EXPORT:GetThree
Creating library Numbers.lib...

A:\>
```

Basic Exports

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /EXPORT:GetOne
         /EXPORT:GetTwo
         /EXPORT:GetThree
Creating library Numbers.lib...

A:\>
```

```
A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetThree
      3 00001010 GetTwo

A:\>
```

Basic Exports

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /EXPORT:GetOne
         /EXPORT:GetTwo
         /EXPORT:GetThree
Creating library Numbers.lib...

A:\>
```

```
A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetThree
      3 00001010 GetTwo

A:\>dumpbin /exports Numbers.lib
ordinal          name
                 GetOne
                 GetThree
                 GetTwo

A:\>
```

Basic Exports

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /EXPORT:GetOne
         /EXPORT:GetTwo
         /EXPORT:GetOnePlusTwo=GetThree
Creating library Numbers.lib...

A:\>
```

```
A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetOnePlusTwo
      3 00001010 GetTwo

A:\>dumpbin /exports Numbers.lib
ordinal          name
                 GetOne
                 GetOnePlusTwo
                 GetTwo

A:\>
```

Renamed Exports

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /EXPORT:GetOne
         /EXPORT:GetTwo
         /EXPORT:GetOnePlusTwo=GetThree
         /EXPORT:GetThree
Creating library Numbers.lib...

A:\>
```

```
A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetOnePlusTwo
      3 00001020 GetThree
      4 00001010 GetTwo

A:\>dumpbin /exports Numbers.lib
ordinal          name
                 GetOne
                 GetOnePlusTwo
                 GetThree
                 GetTwo

A:\>
```

Renamed Exports

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /EXPORT:GetOne
         /EXPORT:GetTwo
         /EXPORT:GetThree,PRIVATE
Creating library Numbers.lib...

A:\>
```

```
A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetThree
      3 00001010 GetTwo

A:\>dumpbin /exports Numbers.lib
ordinal          name
                 GetOne
                 GetTwo

A:\>
```

Private Exports

Module Definition (DEF) Files

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>
```

Module Definition (DEF) Files

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>type Numbers.def
LIBRARY Numbers
EXPORTS
    GetOne
    GetTwo PRIVATE
    GetOnePlusTwo=GetThree

A:\>
```

Module Definition (DEF) Files

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>type Numbers.def
LIBRARY Numbers
EXPORTS
    GetOne
    GetTwo PRIVATE
    GetOnePlusTwo=GetThree

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /DEF:Numbers.def
Creating library Numbers.lib...

A:\>
```

Module Definition (DEF) Files

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>type Numbers.def
LIBRARY Numbers
EXPORTS
    GetOne
    GetTwo PRIVATE
    GetOnePlusTwo=GetThree

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /DEF:Numbers.def
Creating library Numbers.lib...

A:\>
```

```
A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetOnePlusTwo
      3 00001010 GetTwo

A:\>
```

Module Definition (DEF) Files

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

A:\>type Numbers.def
LIBRARY Numbers
EXPORTS
    GetOne
    GetTwo PRIVATE
    GetOnePlusTwo=GetThree

A:\>link Numbers.obj
         /DLL
         /NOENTRY
         /DEF:Numbers.def
Creating library Numbers.lib...

A:\>
```

```
A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetOnePlusTwo
      3 00001010 GetTwo

A:\>dumpbin /exports Numbers.lib
ordinal          name
                 GetOne
                 GetOnePlusTwo

A:\>
```

Module Definition (DEF) Files

A:\>

__declspec(dllexport)

```
A:\>type Numbers.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; }

A:\>
```

__declspec(dllexport)

```
A:\>type Numbers.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; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
Creating library Numbers.lib...

A:\>
```

__declspec(dllexport)

```
A:\>type Numbers.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; }

A:\>link Numbers.obj
         /DLL
         /NOENTRY
Creating library Numbers.lib...

A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetThree
      3 00001010 GetTwo

A:\>
```

__declspec(dllexport)

```
A:\>type Numbers.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; }

A:\>
```

__declspec(dllexport)

```
A:\>type Numbers.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; }

A:\>dumpbin /directives Numbers.obj
Linker Directives
-----------------
/EXPORT:GetOne
/EXPORT:GetTwo
/EXPORT:GetThree

A:\>
```

__declspec(dllexport)

#pragma comment

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

#pragma comment(linker, "/export:GetOne")
#pragma comment(linker, "/export:GetTwo")
#pragma comment(linker, "/export:GetThree")

A:\>
```

#pragma comment

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

#pragma comment(linker, "/export:GetOne")
#pragma comment(linker, "/export:GetTwo")
#pragma comment(linker, "/export:GetThree")

A:\>dumpbin /directives Numbers.obj
Linker Directives
-----------------
/export:GetOne
/export:GetTwo
/export:GetThree

A:\>
```

#pragma comment

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

#pragma comment(linker, "/export:GetOne")
#pragma comment(linker, "/export:GetTwo")
#pragma comment(linker, "/export:GetThree")

A:\>dumpbin /directives Numbers.obj
Linker Directives
-----------------
/export:GetOne
/export:GetTwo
/export:GetThree

A:\>
```

```
A:\>link Numbers.obj
         /DLL
         /NOENTRY
Creating library Numbers.lib...

A:\>
```

#pragma comment

```
A:\>type Numbers.cpp
extern "C" int GetOne()   { return 1; }
extern "C" int GetTwo()   { return 2; }
extern "C" int GetThree() { return 3; }

#pragma comment(linker, "/export:GetOne")
#pragma comment(linker, "/export:GetTwo")
#pragma comment(linker, "/export:GetThree")

A:\>dumpbin /directives Numbers.obj
Linker Directives
-----------------
/export:GetOne
/export:GetTwo
/export:GetThree

A:\>
```

```
A:\>link Numbers.obj
         /DLL
         /NOENTRY
Creating library Numbers.lib...

A:\>dumpbin /exports Numbers.dll
ordinal RVA      name
      1 00001000 GetOne
      2 00001020 GetThree
      3 00001010 GetTwo

A:\>
```

#pragma comment

What Happens When We Load Hello.dll?

The loader needs to…

1. Find 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

What Happens When We Load Hello.dll?

```
#include <Windows.h>

int main()
{
    HMODULE Hello1 = LoadLibraryExW(L"Hello.dll", nullptr, 0); // Hello.dll refcount: 1
    HMODULE Hello2 = LoadLibraryExW(L"Hello.dll", nullptr, 0); // Hello.dll refcount: 2

    // Hello1 and Hello2 will be the same

    FreeLibrary(Hello1); // Hello.dll refcount: 1 (Hello.dll is not unloaded)
    FreeLibrary(Hello2); // Hello.dll refcount: 0 (Hello.dll is unloaded)
}
```

DLLs are Reference Counted

Finding the Right DLL to Load

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

```
HMODULE HelloDll = LoadLibraryExW(L"Hello.dll", nullptr, 0);
```

Finding the Right DLL to Load

```
HMODULE HelloDll = LoadLibraryExW(LR"(A:\Hello.dll)", nullptr, 0);
```

- If A:\Hello.dll has already been loaded, the loader just returns its module handle.
- Otherwise, the loader loads A:\Hello.dll and returns a handle to the newly loaded module.

Note that this means that you can load different DLLs with the same name, using absolute paths:

```
HMODULE HelloDllA = LoadLibraryExW(LR"(A:\Hello.dll)", nullptr, 0);
HMODULE HelloDllB = LoadLibraryExW(LR"(B:\Hello.dll)", nullptr, 0);
```

The “Easy” Case: Absolute Paths

```
HMODULE HelloDll1 = LoadLibraryExW(LR"(A:\Hello.dll)", nullptr, 0);
HMODULE HelloDll2 = LoadLibraryExW(LR"(Hello.dll)",    nullptr, 0);
```

- A Hello.dll has already been loaded in the process, so a handle to that DLL is returned.
- So, HelloDll2 == HelloDll1.

```
HMODULE HelloDllA = LoadLibraryExW(LR"(A:\Hello.dll)", nullptr, 0);
HMODULE HelloDllB = LoadLibraryExW(LR"(B:\Hello.dll)", nullptr, 0);
HMODULE HelloDllX = LoadLibraryExW(LR"(Hello.dll)",    nullptr, 0);
```

- There are two Hello.dlls lodaed in the process; which one should the loader pick?
- It picks the DLL that was loaded first.
- So, HelloDllX == HelloDllA.

Is a DLL with the same name already loaded?

```
HMODULE Kernel32 = LoadLibraryExW(LR"(kernel32.dll)", nullptr, 0);
HMODULE Ntdll    = LoadLibraryExW(LR"(ntdll.dll)",    nullptr, 0);
HMODULE Ole32    = LoadLibraryExW(LR"(ole32.dll)",    nullptr, 0);
```

- These are all known DLLs, so they are loaded from the system directory (e.g., C:\Windows\System32)

```
HMODULE HelloDll2 = LoadLibraryExW(LR"(Hello.dll)", nullptr, 0);
```

- This is not a known DLL, so the loader would continue with the search process

Is the DLL a Known DLL?

```
A:\>reg query "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs
    _wow64        REG_SZ    wow64.dll
    _wow64cpu     REG_SZ    wow64cpu.dll
    _wow64win     REG_SZ    wow64win.dll
    _wowarmhw     REG_SZ    wowarmhw.dll
    advapi32      REG_SZ    advapi32.dll
    clbcatq       REG_SZ    clbcatq.dll
    combase       REG_SZ    combase.dll
    COMDLG32      REG_SZ    COMDLG32.dll
    coml2         REG_SZ    coml2.dll
    DifxApi       REG_SZ    difxapi.dll
    gdi32         REG_SZ    gdi32.dll
    gdiplus       REG_SZ    gdiplus.dll
    IMAGEHLP      REG_SZ    IMAGEHLP.dll
    IMM32         REG_SZ    IMM32.dll
    kernel32      REG_SZ    kernel32.dll
    MSCTF         REG_SZ    MSCTF.dll
    MSVCRT        REG_SZ    MSVCRT.dll
    NORMALIZ      REG_SZ    NORMALIZ.dll
    NSI           REG_SZ    NSI.dll
    ole32         REG_SZ    ole32.dll
    OLEAUT32      REG_SZ    OLEAUT32.dll
    PSAPI         REG_SZ    PSAPI.DLL
    rpcrt4        REG_SZ    rpcrt4.dll
    sechost       REG_SZ    sechost.dll
    Setupapi      REG_SZ    Setupapi.dll
    SHCORE        REG_SZ    SHCORE.dll
    SHELL32       REG_SZ    SHELL32.dll
    SHLWAPI       REG_SZ    SHLWAPI.dll
    user32        REG_SZ    user32.dll
    WLDAP32       REG_SZ    WLDAP32.dll
    WS2_32        REG_SZ    WS2_32.dll
```

Is the DLL a Known DLL?

```
HMODULE HelloDll = LoadLibraryExW(LR"(Hello.dll)", nullptr, 0);
```

The loader will search the following places (assuming standard Windows installation at C:\Windows):

- The directory from which the application loaded: A:\ for our test program
- The system directory: C:\Windows\System32\ or C:\Windows\SysWOW64\
- The 16-bit system directory: C:\Windows\System\
- The Windows directory: C:\Windows\
- The current directory
- The directories listed in the %PATH% environment variable

The current directory used to be searched first (pre-Windows XP SP2)

The Standard Search Order

- DLL Redirection (.local)
- Side-by-Side Components
- %PATH%
- AddDllDirectory
- LoadLibraryEx Flags
  - `LOAD_WITH_ALTERED_SEARCH_PATH`
  - `LOAD_LIBRARY_SEARCH_APPLICATION_DIR`
  - `LOAD_LIBRARY_SEARCH_DEFAULT_DIRS`
  - `LOAD_LIBRARY_SEARCH_DLL_LOADER_DIR`
  - `LOAD_LIBRARY_SEARCH_SYSTEM32`
  - `LOAD_LIBRARY_SEARCH_USER_DIRS`
- Windows Store / Universal Windows Applications (UWAs) are different
- And so on…

The Search Process is Customizable

Mapping the DLL into Memory

```
A:\>dumpbin /headers Hello.dll
OPTIONAL HEADER VALUES
    [...]
    3000 size of image
    [...]

A:\>
```

Hello.dll occupies 0x3000 bytes in memory (that's 12,288 bytes)

How Much Space Does the DLL Require in Memory?

```
A:\>dumpbin /headers Hello.dll
OPTIONAL HEADER VALUES
    [...]
    3000 size of image
    [...]

A:\>dir Hello.dll
                        2,048 Hello.dll
    1 File(s)           2,048 bytes

A:\>
```

Hello.dll occupies 0x3000 bytes in memory (that's 12,288 bytes)

But Hello.dll is only 2,048 bytes

The DLL isn't that big?!

```
A:\>dumpbin /headers Hello.dll
OPTIONAL HEADER VALUES
    [...]
    3000 size of image
    [...]

A:\>dir Hello.dll
                        2,048 Hello.dll
    1 File(s)           2,048 bytes

A:\>dumpbin /headers Hello.dll
OPTIONAL HEADER VALUES
    [...]
    1000 section alignment
     200 file alignment
    [...]

A:\>
```

Hello.dll occupies 0x3000 bytes in memory (that's 12,288 bytes)

But Hello.dll is only 2,048 bytes

4,096 bytes (page size)

512 bytes

The DLL Has Different Alignments on Disk and in Memory

```
A:\>dumpbin /headers Hello.dll
OPTIONAL HEADER VALUES
    [...]
    3000 size of image
    [...]

A:\>dir Hello.dll
                        2,048 Hello.dll
    1 File(s)           2,048 bytes

A:\>dumpbin /headers Hello.dll
OPTIONAL HEADER VALUES
    [...]
    1000 section alignment
     200 file alignment
    [...]

A:\>
```

Hello.dll occupies 0x3000 bytes in memory (that's 12,288 bytes)

But Hello.dll is only 2,048 bytes

4,096 bytes (page size)

512 bytes (FAT sector size)

The DLL Has Different Alignments on Disk and in Memory

Sections Must be Page-Aligned in Memory

```
A:\>dumpbin /headers Hello.dll
SECTION HEADER #1
   .text name
       8 virtual size
    1000 virtual address (70001000 to 70001007)
     200 size of raw data
     400 file pointer to raw data (00000400 to 000005FF)
60000020 flags
         Code
         Execute Read

SECTION HEADER #2
  .rdata name
      D8 virtual size
    2000 virtual address (70002000 to 700020D7)
     200 size of raw data
     600 file pointer to raw data (00000600 to 000007FF)
40000040 flags
         Initialized Data
         Read Only

A:\>
```

Sections Must be Page-Aligned in Memory

We Don’t Need to Store Everything in the File

```
A:\>type HelloBuffer.cpp
char GlobalBuffer[1024 * 1024];

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

A:\>
```

We Don’t Need to Store Everything in the File

```
A:\>type HelloBuffer.cpp
char GlobalBuffer[1024 * 1024];

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

A:\>dir HelloBuffer.dll
                      2,048 HelloBuffer.dll
    1 File(s)         2,048 bytes
    0 Dir(s)  390,011,408,384

A:\>
```

We Don’t Need to Store Everything in the File

```
A:\>type HelloBuffer.cpp
char GlobalBuffer[1024 * 1024];

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

A:\>dir HelloBuffer.dll
                      2,048 HelloBuffer.dll
    1 File(s)         2,048 bytes
    0 Dir(s)  390,011,408,384

A:\>
```

```
A:\>
```

We Don’t Need to Store Everything in the File

```
A:\>type HelloBuffer.cpp
char GlobalBuffer[1024 * 1024];

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

A:\>dir HelloBuffer.dll
                        2,048 HelloBuffer.dll
    1 File(s)           2,048 bytes
    0 Dir(s)  390,011,408,384

A:\>
```

```
A:\>dumpbin /headers HelloBuffer.dll
OPTIONAL HEADER VALUES
    [...]
    103000 size of image
    [...]

SECTION HEADER #3
   .data name
  100000 virtual size
    3000 virtual address (70003000 to 70102FFF)
       0 size of raw data
       0 file pointer to raw data
C0000040 flags
         Initialized Data
         Read Write

A:\>
```

We Don’t Need to Store Everything in the File

To Map a DLL into Memory, the Loader Needs to…

- open the DLL file and read the image size,
- allocate a contiguous, page-aligned block of memory of that size, and
- copy the contents of each section into the appropriate area of that block of memory.

Later, it will set the appropriate page protections on each page of the mapped DLL.

To Map a DLL into Memory, the Loader Needs to…

Relocation

A:\>

PointerToTwo => Two

```
A:\>type PointerGlobal.cpp
extern "C" __declspec(dllexport) size_t        const Two          = 2;
extern "C" __declspec(dllexport) size_t const* const PointerToTwo = &Two;

A:\>
```

PointerToTwo => Two

```
A:\>type PointerGlobal.cpp
extern "C" __declspec(dllexport) size_t        const Two          = 2;
extern "C" __declspec(dllexport) size_t const* const PointerToTwo = &Two;

A:\>link PointerGlobal.obj
         /DLL
         /OUT:PointerGlobal.dll
         /NOENTRY
Creating library PointerGlobal.lib...

A:\>
```

PointerToTwo => Two

How Do These Look in the DLL?

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>
```

How Do These Look in the DLL?

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>dumpbin /rawdata:8 /section:.rdata PointerGlobal.dll
  0000000070001000: 0000000000000002 0000000070001000   ............p....
  0000000070001010: 59C4181E00000000 0000000D00000000   ......ÄY........
  [...continued...]

A:\>
```

How Do These Look in the DLL?

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>dumpbin /rawdata:8 /section:.rdata PointerGlobal.dll
  0000000070001000: 0000000000000002 0000000070001000   ............p....
  0000000070001010: 59C4181E00000000 0000000D00000000   ......ÄY........
  [...continued...]

A:\>
```

Two

How Do These Look in the DLL?

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>dumpbin /rawdata:8 /section:.rdata PointerGlobal.dll
  0000000070001000: 0000000000000002 0000000070001000   ............p....
  0000000070001010: 59C4181E00000000 0000000D00000000   ......ÄY........
  [...continued...]

A:\>
```

Two

PointerToTwo

How Do These Look in the DLL?

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>dumpbin /rawdata:8 /section:.rdata PointerGlobal.dll
  0000000070001000: 0000000000000002 0000000070001000   ............p....
  0000000070001010: 59C4181E00000000 0000000D00000000   ......ÄY........
  [...continued...]

A:\>
```

Two

PointerToTwo

This only works if PointerGlobal.dll is loaded at 0x7000’0000

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>dumpbin /rawdata:8 /section:.rdata PointerGlobal.dll
  0000000070001000: 0000000000000002 0000000070001000   ............p....
  0000000070001010: 59C4181E00000000 0000000D00000000   ......ÄY........
  [...continued...]
```

Two PointerToTwo

```
A:\>dumpbin /relocations pointerglobal.dll
BASE RELOCATIONS #2
    1000 RVA,        C SizeOfBlock
       8  DIR64      0000000070001000
       0  ABS

A:\>
```

There is a pointer located 8 bytes…

…from the start of the section beginning at RVA 0x1000

Relocation to the Rescue

The loader will update each pointer listed in the relocation table by updating it via:

```
Pointer = Pointer
            - PreferredBaseAddress
            + ActualBaseAddress;
```

Relocation to the Rescue

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>dumpbin /rawdata:8 /section:.rdata PointerGlobal.dll
  0000000070001000: 0000000000000002 0000000070001000   ............p....
  0000000070001010: 59C4181E00000000 0000000D00000000   ......ÄY........
  [...continued...]

A:\>
```

```
  00000000'70001000  (Original Pointer Value)
- 00000000'70000000  (Preferred Base Address)
+ 00000000'90000000  (Actual Base Address)
= 00000000'90001000  (New Pointer Value)
```

So, if PointersGlobal.dll was loaded at 9000'0000…

```
A:\>dumpbin /exports PointerGlobal.dll
ordinal hint RVA      name
      1    0 00001008 PointerToTwo
      2    1 00001000 Two

A:\>dumpbin /rawdata:8 /section:.rdata PointerGlobal.dll
  0000000070001000: 0000000000000002 0000000090001000   ............p....
  0000000070001010: 59C4181E00000000 0000000000000000   ......ÄY........
  [...continued...]

A:\>
```

```
  00000000'70001000  (Original Pointer Value)
- 00000000'70000000  (Preferred Base Address)
+ 00000000'90000000  (Actual Base Address)
= 00000000'90001000  (New Pointer Value)
```

So, if PointersGlobal.dll was loaded at 9000'0000…

Loading Dependencies and Binding Imports

A:\>

Hello.dll Doesn’t Have Any Dependencies

```
A:\>dumpbin /imports Hello.dll

A:\>
```

Hello.dll Doesn’t Have Any Dependencies

Let’s Give Hello.dll a Dependency

```
A:\>type Hello.cpp
#include <Windows.h>

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

extern "C" void __cdecl GetWideGreeting(wchar_t* buffer, int size)
{
    MultiByteToWideChar(CP_UTF8, 0, GetGreeting(), -1, buffer, size);
}

A:\>
```

Let’s Give Hello.dll a Dependency

Let’s Give Hello.dll a Dependency

```
A:\>link Hello.obj
         /DLL
         /NOENTRY
         /EXPORT:GetGreeting
         /EXPORT:GetWideGreeting
         kernel32.lib
Creating library Hello.lib...

A:\>
```

Let’s Give Hello.dll a Dependency

```
A:\>link Hello.obj
         /DLL
         /NOENTRY
         /EXPORT:GetGreeting
         /EXPORT:GetWideGreeting
         kernel32.lib
Creating library Hello.lib...

A:\>dumpbin /exports Hello.dll
Section contains the following exports for Hello.dll
    ordinal hint RVA      name
          1    0 00001000 GetGreeting
          2    1 00001010 GetWideGreeting

A:\>
```

Let’s Give Hello.dll a Dependency

```
A:\>link Hello.obj
         /DLL
         /NOENTRY
         /EXPORT:GetGreeting
         /EXPORT:GetWideGreeting
         kernel32.lib
Creating library Hello.lib...

A:\>dumpbin /exports Hello.dll
Section contains the following exports for Hello.dll
    ordinal hint RVA      name
          1    0 00001000 GetGreeting
          2    1 00001010 GetWideGreeting

A:\>dumpbin /imports Hello.dll
Section contains the following imports:
    KERNEL32.dll
        3E5 MultiByteToWideChar

A:\>
```

Let’s Give Hello.dll a Dependency

```
// For exposition only...
for (auto& DllDependency : DllDependencies)
{
    DllDependency.Handle = LoadLibraryExW(DllDependency.Name, nullptr, 0);
    if (DllDependency.Handle == nullptr)
    {
        // Return failure
    }

    for (auto& Import : DllDependency.Imports)
    {
        Import.Address = GetProcAddress(DllDependency.Handle, Import.Name);
        if (Import.Address == nullptr)
        {
            // Return failure
        }
    }
}
```

The Dependency Loading and Binding Process

Or, in English…

- For each DLL dependency,
  - Load the DLL…
  - …then get all of the required exports

The Dependency Loading and Binding Process

Initializing the DLL

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

The **instance** is a handle to the DLL—the same one that will be returned from LoadLibrary.

The **reason** indicates why the loader 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

The **reserved** parameter gives a little extra information:

- For process attach: It is null if DLL loaded via LoadLibrary; non-null if loaded as implicit dependency
- For process detach: It is null if DLL unloaded via FreeLibrary; non-null if the process is terminating

Returns TRUE on success; FALSE on failure.

Calls to DllMain are synchronized by a global lock, called the Loader Lock

The DLL Entry Point (or “DllMain”)

Not All DLLs Have an Entry Point

```
A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>
```

Not All DLLs Have an Entry Point

```
A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>link Hello.obj
         /DLL
         /NOENTRY
         /NODEFAULTLIB
         /EXPORT:GetGreeting
Creating library Hello.lib...

A:\>
```

We told the linker not to give Hello.dll an entry point…

Not All DLLs Have an Entry Point

```
A:\>type Hello.cpp
extern "C" char const* __cdecl GetGreeting()
{
    return "Hello, C++ Programmers!";
}

A:\>link Hello.obj
         /DLL
         /NOENTRY
         /NODEFAULTLIB
         /EXPORT:GetGreeting
Creating library Hello.lib...

A:\>dumpbin /headers Hello.dll
OPTIONAL HEADER VALUES
    [...]
    0 entry point
    [...]

A:\>
```

We told the linker not to give Hello.dll an entry point…

…and we can verify with dumpbin that Hello.dll doesn't have one.

Not All DLLs Have an Entry Point

Let’s Build a DLL with an Entry Point…

```
A:\>type DllWithEntryPoint.cpp
#include <stdio.h>
#include <Windows.h>

extern "C" BOOL WINAPI DllMain(HINSTANCE instance, DWORD reason, LPVOID reserved)
{
    switch (reason)
    {
    case DLL_PROCESS_ATTACH: puts("DllMain called for DLL_PROCESS_ATTACH"); break;
    case DLL_PROCESS_DETACH: puts("DllMain called for DLL_PROCESS_DETACH"); break;
    }

    return TRUE;
}

A:\>
```

Let’s Build a DLL with an Entry Point…

```
A:\>type DllWithEntryPoint.cpp
#include <stdio.h>
#include <Windows.h>

extern "C" BOOL WINAPI DllMain(HINSTANCE instance, DWORD reason, LPVOID reserved)
{
    switch (reason)
    {
    case DLL_PROCESS_ATTACH: puts("DllMain called for DLL_PROCESS_ATTACH"); break;
    case DLL_PROCESS_DETACH: puts("DllMain called for DLL_PROCESS_DETACH"); break;
    }

    return TRUE;
}

A:\>link DllWithEntryPoint.obj
         /DLL
         /ENTRY:DllMain

A:\>
```

Let’s Build a DLL with an Entry Point…

A:\>

…and a Test Program to Demonstrate It

```
A:\>type TestEntryPoint.cpp
#include <stdio.h>
#include <Windows.h>

int main()
{
    printf("About to load DLL...\n");
    HMODULE const TestDll = LoadLibraryExW(L"DllWithEntryPoint.dll", nullptr, 0);
    printf("DLL loaded.  About to unload DLL...\n");
    FreeLibrary(TestDll);
    printf("DLL unloaded.\n");
}

A:\>
```

…and a Test Program to Demonstrate It

```
A:\>type TestEntryPoint.cpp
#include <stdio.h>
#include <Windows.h>

int main()
{
    printf("About to load DLL...\n");
    HMODULE const TestDll = LoadLibraryExW(L"DllWithEntryPoint.dll", nullptr, 0);
    printf("DLL loaded.  About to unload DLL...\n");
    FreeLibrary(TestDll);
    printf("DLL unloaded.\n");
}

A:\>link TestEntryPoint.obj

A:\>
```

```
A:\>TestEntryPoint.exe
About to load DLL...
DllMain called for DLL_PROCESS_ATTACH
DLL loaded.  About to unload DLL...
DllMain called for DLL_PROCESS_DETACH
DLL unloaded.
A:\>
```

...and a Test Program to Demonstrate It

MSDN has an article on “Dynamic-Link Library Best Practices”

- https://msdn.microsoft.com/en-us/library/windows/desktop/dn633971.aspx

The short version is:

- Do as little as possible in your entry point
- Be *very* careful when calling into other DLLs from your entry point
- Do not synchronize with other threads from your entry point

In general, in C and C++ programs, you won’t specify your own entry point

- We’ll look at this in a little bit when we discuss C++-specific things for DLLs

Be Careful with Your Entry Point...

Diagnosing DLL Load Failures

A:\>

Loader Snaps

```
A:\>del Hello.dll

A:\>
```

Loader Snaps

```
A:\>del Hello.dll

A:\>PrintGreeting.exe

A:\>
```

Loader Snaps

```
A:\>del Hello.dll

A:\>PrintGreeting.exe

A:\>echo %errorlevel%
-1073741819

A:\>
```

Loader Snaps

```
A:\>del Hello.dll

A:\>PrintGreeting.exe

A:\>echo %errorlevel%
-1073741819

A:\>gflags /i PrintGreeting.exe +sls

A:\>
```

Show Loader Snaps

Loader Snaps

```
A:\>del Hello.dll

A:\>PrintGreeting.exe

A:\>echo %errorlevel%
-1073741819

A:\>gflags /i PrintGreeting.exe +sls

A:\>cdb PrintGreeting.exe
[...]
LdrpInitializeProcess - INFO: Beginning execution of PrintGreeting.exe (A:\PrintGreeting.exe)
        Current directory: A:\
        Package directories: (null)
[...]
```

Show Loader Snaps

Loader Snaps

```
LdrLoadDll - ENTER: DLL name: Hello.dll
LdrpLoadDllInternal - ENTER: DLL name: Hello.dll
LdrpFindKnownDll - ENTER: DLL name: Hello.dll
LdrpFindKnownDll - RETURN: Status: 0xc0000135
LdrpSearchPath - ENTER: DLL name: Hello.dll
LdrpComputeLazyDllPath - INFO: DLL search path computed:
    A:\;
    C:\Windows\SYSTEM32;
    C:\Windows\system;
    C:\Windows;
    .;
    C:\Debuggers;
    C:\Program Files (x86)\Microsoft Visual Studio\2017\[etc.]
```

Loader Snaps

```
LdrpResolveDllName - ENTER: DLL name: A:\Hello.dll
LdrpResolveDllName - RETURN: Status: 0xc0000135
LdrpResolveDllName - ENTER: DLL name: C:\Windows\SYSTEM32\Hello.dll
LdrpResolveDllName - RETURN: Status: 0xc0000135
LdrpResolveDllName - ENTER: DLL name: C:\Windows\system\Hello.dll
LdrpResolveDllName - RETURN: Status: 0xc0000135
LdrpResolveDllName - ENTER: DLL name: C:\Windows\Hello.dll
LdrpResolveDllName - RETURN: Status: 0xc0000135
LdrpResolveDllName - ENTER: DLL name: .\Hello.dll
LdrpResolveDllName - RETURN: Status: 0xc0000135
LdrpResolveDllName - ENTER: DLL name: C:\Debuggers\Hello.dll
LdrpResolveDllName - RETURN: Status: 0xc0000135
[...it checks the rest of the path...]
LdrpProcessWork - ERROR: Unable to load DLL: "Hello.dll",
    Parent Module: "(null)",
    Status: 0xc0000135
LdrpLoadDllInternal - RETURN: Status: 0xc0000135
LdrLoadDll - RETURN: Status: 0xc0000135
```

STATUS_DLL_NOT_FOUND

Loader Snaps

```
A:\>link Hello.obj
         /DLL
         /NOENTRY
         /NODEFAULTLIB
         /EXPORT:GetGreeting
```

```
LdrpResolveDllName - ENTER: DLL name: A:\Hello.dll
LdrpResolveDllName - RETURN: Status: 0x00000000
LdrpSearchPath - RETURN: Status: 0x00000000
LdrpMinimalMapModule - ENTER: DLL name: A:\Hello.dll
LdrpMinimalMapModule - RETURN: Status: 0x00000000
LdrpInitializeNode - INFO: Calling init routine 0000000000000000 for DLL "A:\Hello.dll"
LdrpLoadDllInternal - RETURN: Status: 0x00000000
LdrLoadDll - RETURN: Status: 0x00000000
LdrpReportError - WARNING: Locating export "GetGreeting" for DLL "Unknown" failed with
    status: 0xc0000139.
```

STATUS_ENTRY_POINT_NOT_FOUND

Loader Snaps

```
A:\>gflags /i PrintGreeting.exe -sls
```

Loader Snaps

Output

Show output from: Debug

```
6bfc:7db8 @ 242071250 - LdrLoadDll - ENTER: DLL name: Hello.dll
6bfc:7db8 @ 242071250 - LdrpLoadDllInternal - ENTER: DLL name: Hello.dll
6bfc:7db8 @ 242071250 - LdrpFindKnownDll - ENTER: DLL name: Hello.dll
6bfc:7db8 @ 242071250 - LdrpFindKnownDll - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpSearchPath - ENTER: DLL name: Hello.dll
6bfc:7db8 @ 242071250 - LdrpComputeLazyDllPath - INFO: DLL search path computed: A:\;C:\Windows\SY
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: A:\Hello.dll
6bfc:7db8 @ 242071250 - LdrpResolveDllName - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: C:\Windows\SYSTEM32\Hello.dll
6bfc:7db8 @ 242071250 - LdrpResolveDllName - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: C:\Windows\system\Hello.dll
6bfc:7db8 @ 242071250 - LdrpResolveDllName - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: C:\Windows\Hello.dll
6bfc:7db8 @ 242071250 - LdrpResolveDllName - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: .\Hello.dll
6bfc:7db8 @ 242071250 - LdrpResolveDllName - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: C:\Windows\system32\Hello.dll
6bfc:7db8 @ 242071250 - LdrpResolveDllName - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: C:\Windows\Hello.dll
6bfc:7db8 @ 242071250 - LdrpResolveDllName - RETURN: Status: 0xc0000135
6bfc:7db8 @ 242071250 - LdrpResolveDllName - ENTER: DLL name: C:\Windows\System32\Wbem\Hello.dll
```

Call Stack | Breakpoints | Exception Settings | Command Window | Immediate Window | Output

__declspec(dllimport)

```
A:\>type Numbers.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; }

A:\>
```

We Already Saw __declspec(dllexport)

```
A:\>type NumbersCaller.cpp
extern "C" __declspec(dllimport) int GetOne();
extern "C" __declspec(dllimport) int GetTwo();
extern "C" __declspec(dllimport) int GetThree();

A:\>
```

There Is Also a __declspec(dllimport)

```
A:\>type NumbersCaller.cpp
extern "C" int GetOne();
extern "C" int GetTwo();
extern "C" int GetThree();

int main()
{
    return GetOne();
}

A:\>
```

A Caller without __declspec(dllimport)

```
A:\>type NumbersCaller.cpp
extern "C" int GetOne();
extern "C" int GetTwo();
extern "C" int GetThree();

int main()
{
    return GetOne();
}

A:\>link NumbersCaller.obj
         Numbers.lib

A:\>
```

A Caller without __declspec(dllimport)

```
A:\>type NumbersCaller.cpp
extern "C" int GetOne();
extern "C" int GetTwo();
extern "C" int GetThree();

int main()
{
    return GetOne();
}

A:\>link NumbersCaller.obj
         Numbers.lib

A:\>
```

```
// If we look at the disassembly, we'll see:

main():

    call GetOne


GetOne():

    jmp  qword ptr[__imp_GetOne]
```

A Caller without __declspec(dllimport)

```
A:\>type NumbersCaller.cpp
extern "C" __declspec(dllimport) int GetOne();
extern "C" __declspec(dllimport) int GetTwo();
extern "C" __declspec(dllimport) int GetThree();

int main()
{
    return GetOne();
}

A:\>
```

A Caller *with* __declspec(dllimport)

```
A:\>type NumbersCaller.cpp
extern "C" __declspec(dllimport) int GetOne();
extern "C" __declspec(dllimport) int GetTwo();
extern "C" __declspec(dllimport) int GetThree();

int main()
{
    return GetOne();
}

A:\>link NumbersCaller.obj
         Numbers.lib

A:\>
```

A Caller *with* __declspec(dllimport)

```
A:\>type NumbersCaller.cpp
extern "C" __declspec(dllimport) int GetOne();
extern "C" __declspec(dllimport) int GetTwo();
extern "C" __declspec(dllimport) int GetThree();

int main()
{
    return GetOne();
}

A:\>link NumbersCaller.obj
         Numbers.lib

A:\>
```

```
// If we look at the disassembly, we'll see:

main():

    call qword ptr [__imp_GetOne]
```

A Caller *with* __declspec(dllimport)

Exporting Data

A:\>

Exporting Data

```
A:\>type Constants.cpp
extern "C" int const One = 1;
extern "C" int const Two = 2;

A:\>
```

Exporting Data

```
A:\>type Constants.cpp
extern "C" int const One = 1;
extern "C" int const Two = 2;

A:\>link Constants.obj
         /dll
         /out:Constants.dll
         /noentry
         /nodefaultlib
         /export:One,DATA
         /export:Two,DATA
Creating library Constants.lib...

A:\>
```

Exporting Data

A:\>

Exporting Data

```
A:\>dumpbin /exports Constants.dll

  Section contains the following exports for Constants.dll

    ordinal hint RVA      name

          1    0 00001000 One
          2    1 00001004 Two

A:\>
```

Exporting Data

```
A:\>dumpbin /exports Constants.dll

  Section contains the following exports for Constants.dll

    ordinal hint RVA      name

          1    0 00001000 One
          2    1 00001004 Two

A:\>dumpbin /rawdata:4 /section:.rdata Constants.dll
RAW DATA #1
  70001000: 00000001 00000002 00000000 00000000  ................

A:\>
```

Exporting Data

```
A:\>dumpbin /exports Constants.dll

  Section contains the following exports for Constants.dll

    ordinal hint RVA      name

          1    0 00001000 One
          2    1 00001004 Two

A:\>dumpbin /rawdata:4 /section:.rdata Constants.dll
RAW DATA #1
  70001000: 00000001 00000002 00000000 00000000  ................

A:\>
```

One

Exporting Data

```
A:\>dumpbin /exports Constants.dll

  Section contains the following exports for Constants.dll

    ordinal hint RVA      name

          1    0 00001000 One
          2    1 00001004 Two

A:\>dumpbin /rawdata:4 /section:.rdata Constants.dll
RAW DATA #1
  70001000: 00000001 00000002 00000000 00000000  ................

A:\>
```

One Two

Exporting Data

A:\>

Importing Data via Import Library

```
A:\>type UseConstants.cpp
#include <stdio.h>

extern "C" int One;
extern "C" int Two;

int main()
{
    printf("One is %d; Two is %d\n", One, Two);
}

A:\>
```

Importing Data via Import Library

```
A:\>type UseConstants.cpp
#include <stdio.h>

extern "C" int One;
extern "C" int Two;

int main()
{
    printf("One is %d; Two is %d\n", One, Two);
}

A:\>cl UseConstants.cpp /link Constants.lib
UseConstants.cpp

UseConstants.obj : error LNK2019: unresolved external symbol One referenced in function main
UseConstants.obj : error LNK2019: unresolved external symbol Two referenced in function main
UseConstants.exe : fatal error LNK1120: 2 unresolved externals

A:\>
```

Importing Data via Import Library

__declspec(dllimport) Is Required for Data Imports

```
A:\>type UseConstants.cpp
#include <stdio.h>

extern "C" __declspec(dllimport) int One;
extern "C" __declspec(dllimport) int Two;

int main()
{
    printf("One is %d; Two is %d\n", One, Two);
}

A:\>
```

__declspec(dllimport) is required for data imports

__declspec(dllimport) Is Required for Data Imports

```
A:\>type UseConstants.cpp
#include <stdio.h>

extern "C" __declspec(dllimport) int One;
extern "C" __declspec(dllimport) int Two;

int main()
{
    printf("One is %d; Two is %d\n", One, Two);
}

A:\>cl UseConstants.cpp /link Constants.lib
UseConstants.cpp

A:\>
```

__declspec(dllimport) is required for data imports

__declspec(dllimport) Is Required for Data Imports

```
A:\>type UseConstants.cpp
#include <stdio.h>

extern "C" __declspec(dllimport) int One;
extern "C" __declspec(dllimport) int Two;

int main()
{
    printf("One is %d; Two is %d\n", One, Two);
}

A:\>cl UseConstants.cpp /link Constants.lib
UseConstants.cpp

A:\>UseConstants.exe
One is 1; Two is 2

A:\>
```

__declspec(dllimport) is required for data imports

__declspec(dllimport) Is Required for Data Imports

```
#include <stdio.h>
#include <Windows.h>

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

    using ConstantPointer = int const*;

    ConstantPointer const One =
        reinterpret_cast<ConstantPointer>(
            GetProcAddress(ConstantsDll, "One"));

    ConstantPointer const Two =
        reinterpret_cast<ConstantPointer>(
            GetProcAddress(ConstantsDll, "Two"));

    printf("One is %d; Two is %d\n", *One, *Two);

    FreeLibrary(ConstantsDll); // Should use RAII
}
```

Note that the export is an int const*, not an int!

Importing Data via GetProcAddress

Delay Loading

```
A:\>type DllWithEntryPoint.cpp
#include <stdio.h>
#include <Windows.h>

extern "C" BOOL WINAPI DllMain(HINSTANCE instance, DWORD reason, LPVOID reserved)
{
    switch (reason)
    {
    case DLL_PROCESS_ATTACH: puts("DllMain called for DLL_PROCESS_ATTACH"); break;
    case DLL_PROCESS_DETACH: puts("DllMain called for DLL_PROCESS_DETACH"); break;
    }

    return TRUE;
}

extern "C" int GetZero()
{
    return 0;
}
```

Delay Loading

Delay Loading

```
A:\>link DllWithEntryPoint.obj
         /DLL
         /ENTRY:DllMain
         /EXPORT:GetZero
Creating library DllWithEntryPoint.lib

A:\>
```

Delay Loading

Delay Loading

```
A:\>type TestDelayLoad.cpp
#include <stdio.h>

int main()
{
    puts("main():  Before call to GetZero()");
    GetZero();
    puts("main():  After call to GetZero()");
}

A:\>
```

Delay Loading

Delay Loading

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib



A:\>
```

Delay Loading

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib



A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    DllWithEntryPoint.dll
    KERNEL32.dll



A:\>
```

Delay Loading

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib
```

```
A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    DllWithEntryPoint.dll
    KERNEL32.dll
```

```
A:\>TestDelayLoad.exe
DllMain called for DLL_PROCESS_ATTACH
main():  Before call to GetZero()
main():  After call to GetZero()
DllMain called for DLL_PROCESS_DETACH

A:\>
```

Delay Loading

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib
```

```
A:\>
```

```
A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    DllWithEntryPoint.dll
    KERNEL32.dll
```

```
A:\>TestDelayLoad.exe
DllMain called for DLL_PROCESS_ATTACH
main():  Before call to GetZero()
main():  After call to GetZero()
DllMain called for DLL_PROCESS_DETACH

A:\>
```

Delay Loading

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib
```

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib
         /DELAYLOAD:DllWithEntryPoint.dll
         delayimp.lib

A:\>
```

```
A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    DllWithEntryPoint.dll
    KERNEL32.dll
```

```
A:\>TestDelayLoad.exe
DllMain called for DLL_PROCESS_ATTACH
main():  Before call to GetZero()
main():  After call to GetZero()
DllMain called for DLL_PROCESS_DETACH

A:\>
```

Delay Loading

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib


A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    DllWithEntryPoint.dll
    KERNEL32.dll


A:\>TestDelayLoad.exe
DllMain called for DLL_PROCESS_ATTACH
main():  Before call to GetZero()
main():  After call to GetZero()
DllMain called for DLL_PROCESS_DETACH

A:\>
```

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib
         /DELAYLOAD:DllWithEntryPoint.dll
         delayimp.lib

A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    KERNEL32.dll

Image has the following delay load
dependencies:
    DllWithEntryPoint.dll

A:\>
```

Delay Loading

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib

A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    DllWithEntryPoint.dll
    KERNEL32.dll

A:\>TestDelayLoad.exe
DllMain called for DLL_PROCESS_ATTACH
main():  Before call to GetZero()
main():  After call to GetZero()
DllMain called for DLL_PROCESS_DETACH

A:\>
```

```
A:\>link TestDelayLoad.obj
         DllWithEntryPoint.lib
         /DELAYLOAD:DllWithEntryPoint.dll
         delayimp.lib

A:\>dumpbin /dependents TestDelayLoad.exe
Image has the following dependencies:
    KERNEL32.dll

Image has the following delay load
dependencies:
    DllWithEntryPoint.dll

A:\>TestDelayLoad.exe
main():  Before call to GetZero()
DllMain called for DLL_PROCESS_ATTACH
main():  After call to GetZero()
DllMain called for DLL_PROCESS_DETACH

A:\>
```

Delay Loading

```
void* __imp_GetZero = __imp_load_GetZero;

void __imp_load_GetZero()
{
    return __tailMerge_DllWithEntryPoint_dll(&__imp_GetZero);
}
```

__tailMerge_DllWithEntryPoint_dll does the following:

- Push parameter registers onto the stack
- Call __delayLoadHelper2, which will:
  - HANDLE DllWithEntryPointHandle = LoadLibrary(DllWithEntryPoint.dll)
  - __imp_GetZero = GetProcAddress(DllWithEntryPointHandle, “GetZero”)
- Pop parameter registers back off of the stack
- Jump to __imp_GetZero (i.e., transfer control to the now-imported function)

If the LoadLibrary or GetProcAddress fails, a structured exception is raised

- You can also hook failures, so you can customize the way the delay-loaded DLL is loaded, if necessary

Delay Loading

C++ and DLLs

```
#include <stdio.h>
#include <Windows.h>

struct Squawker
{
    Squawker()  { puts("Constructed Squawker"); }
    ~Squawker() { puts("Destroyed Squawker");   }
};

Squawker AGlobalVariable;

extern "C" BOOL WINAPI DllMain(HINSTANCE instance, DWORD reason, LPVOID reserved)
{
    switch (reason)
    {
    case DLL_PROCESS_ATTACH: puts("DllMain called for DLL_PROCESS_ATTACH"); break;
    case DLL_PROCESS_DETACH: puts("DllMain called for DLL_PROCESS_DETACH"); break;
    }

    return TRUE;
}
```

Global Variables in DLLs

A:\>

Global Variables in DLLs

```
A:\>link Squawker.obj /DLL /ENTRY:DllMain

A:\>
```

Global Variables in DLLs

```
A:\>link Squawker.obj /DLL /ENTRY:DllMain

A:\>type TestSquawker.cpp
#include <stdio.h>
#include <Windows.h>

int main()
{
    puts("About to load DLL...");
    HMODULE SquawkerDll = LoadLibraryExW(L"Squawker.dll", nullptr, 0);
    puts("DLL loaded.  About to unload DLL...");
    FreeLibrary(SquawkerDll);
    puts("DLL unloaded.");
}

A:\>
```

Global Variables in DLLs

```
A:\>link Squawker.obj /DLL /ENTRY:DllMain

A:\>type TestSquawker.cpp
#include <stdio.h>
#include <Windows.h>

int main()
{
    puts("About to load DLL...");
    HMODULE SquawkerDll = LoadLibraryExW(L"Squawker.dll", nullptr, 0);
    puts("DLL loaded.  About to unload DLL...");
    FreeLibrary(SquawkerDll);
    puts("DLL unloaded.");
}

A:\>link TestSquawker.obj

A:\>
```

Global Variables in DLLs

```
A:\>link Squawker.obj /DLL /ENTRY:DllMain

A:\>type TestSquawker.cpp
#include <stdio.h>
#include <Windows.h>

int main()
{
    puts("About to load DLL...");
    HMODULE SquawkerDll = LoadLibraryExW(L"Squawker.dll", nullptr, 0);
    puts("DLL loaded.  About to unload DLL...");
    FreeLibrary(SquawkerDll);
    puts("DLL unloaded.");
}

A:\>link TestSquawker.obj

A:\>
```

```
A:\>TestSquawker.exe
About to load DLL...
DllMain called for DLL_PROCESS_ATTACH
DLL loaded.  About to unload DLL...
DllMain called for DLL_PROCESS_DETACH
DLL unloaded.
```

Global Variables in DLLs

```
A:\>link Squawker.obj /DLL /ENTRY:DllMain

A:\>type TestSquawker.cpp
#include <stdio.h>
#include <Windows.h>

int main()
{
    puts("About to load DLL...");
    HMODULE SquawkerDll = LoadLibraryExW(L"Squawker.dll", nullptr, 0);
    puts("DLL loaded.  About to unload DLL...");
    FreeLibrary(SquawkerDll);
    puts("DLL unloaded.");
}

A:\>link TestSquawker.obj

A:\>
```

Use Default Entry Point (_DllMainCRTStartup)

```
A:\>TestSquawker.exe
About to load DLL...
Constructed Squawker
DllMain called for DLL_PROCESS_ATTACH
DLL loaded.  About to unload DLL...
DllMain called for DLL_PROCESS_DETACH
Destroyed Squawker
DLL unloaded.
```

The C Runtime (CRT) provides an entry point that C and C++ programs should use

- _DllMainCRTStartup
- This is the default entry point that the linker will pick if you don't specify /ENTRY
- It handles initialization of C and C++ language feature support in the DLL

At DLL_PROCESS_ATTACH:

- If the CRT is statically linked into the DLL, initializes the statically linked CRT
- Initializes security cookie and other run-time check support
- Runs constructors for global variables
- Initializes atexit() support within the DLL
- Calls user-defined DllMain if one is defined

At DLL_PROCESS_DETACH:

- Calls atexit()-registered functions
- Runs destructors for global variables
- If the CRT is statically linked into the DLL, shuts down the statically linked CRT

Note that DLLs are independent with respect to global variables and atexit()

_DllMainCRTStartup

Exporting C++ Functions

```
A:\>type Adder.cpp
__declspec(dllexport) int    Add(int    x, int    y) { return x + y; }
__declspec(dllexport) double Add(double x, double y) { return x + y; }

A:\>
```

Exporting C++ Functions

```
A:\>type Adder.cpp
__declspec(dllexport) int    Add(int    x, int    y) { return x + y; }
__declspec(dllexport) double Add(double x, double y) { return x + y; }

A:\>link Adder.obj /dll
Creating library Adder.lib...

A:\>
```

Exporting C++ Functions

```
A:\>type Adder.cpp
__declspec(dllexport) int    Add(int    x, int    y) { return x + y; }
__declspec(dllexport) double Add(double x, double y) { return x + y; }

A:\>link Adder.obj /dll
Creating library Adder.lib...

A:\>dumpbin /exports Adder.dll
Section contains the following exports for Adder.dll
    ordinal hint RVA      name
          1    0 00001000 ?Add@@YAHHH@Z
          2    1 00001020 ?Add@@YANNN@Z

A:\>
```

Exporting C++ Functions

Exporting C++ Classes

```
A:\>type Counter.cpp
class __declspec(dllexport) Counter
{
public:
    Counter()
        : Value(0)
    { }
    Counter(int InitialValue)
        : Value(InitialValue)
    { }

    int  GetValue()  { return Value; }
    void Increment() { ++Value;      }

private:
    int Value;
};
```

Exporting C++ Classes

```
A:\>type Counter.cpp
class __declspec(dllexport) Counter
{
public:
    Counter()
        : Value(0)
    { }
    Counter(int InitialValue)
        : Value(InitialValue)
    { }

    int  GetValue()  { return Value; }
    void Increment() { ++Value;      }

private:
    int Value;
};
```

```
A:\>
```

Exporting C++ Classes

```
A:\>type Counter.cpp
class __declspec(dllexport) Counter
{
public:
    Counter()
        : Value(0)
    { }
    Counter(int InitialValue)
        : Value(InitialValue)
    { }

    int  GetValue()  { return Value; }
    void Increment() { ++Value;      }

private:
    int Value;
};
```

```
A:\>link Counter.obj /dll

A:\>
```

Exporting C++ Classes

```
A:\>type Counter.cpp
class __declspec(dllexport) Counter
{
public:
    Counter()
        : Value(0)
    { }
    Counter(int InitialValue)
        : Value(InitialValue)
    { }

    int  GetValue()  { return Value; }
    void Increment() { ++Value;      }

private:
    int Value;
};
```

```
A:\>link Counter.obj /dll

A:\>dumpbin /exports Counter.dll
Dump of file Counter.dll
RVA      name
00001000 ??0Counter@@QEAA@H@Z
00001020 ??0Counter@@QEAA@XZ
00001040 ??4Counter@@QEAAEAV0@$$QEAV0@@Z
00001060 ??4Counter@@QEAAEAV0@AEBV0@@Z
00001080 ?GetValue@Counter@@QEAAHXZ
00001090 ?Increment@Counter@@QEAAXXZ

A:\>
```

Exporting C++ Classes

Don’t.

Advice for Exporting C++ Functions and Classes

Threads and Thread-Local Storage

```
#include <Windows.h>

extern "C" BOOL WINAPI DllMain(HINSTANCE instance, DWORD reason, LPVOID reserved)
{
    switch (reason)
    {
    case DLL_PROCESS_ATTACH: puts("DllMain called for DLL_PROCESS_ATTACH"); break;
    case DLL_PROCESS_DETACH: puts("DllMain called for DLL_PROCESS_DETACH"); break;

    case DLL_THREAD_ATTACH:  puts("DllMain called for DLL_THREAD_ATTACH" ); break;
    case DLL_THREAD_DETACH:  puts("DllMain called for DLL_THREAD_DETACH" ); break;
    }

    return TRUE;
}
```

DLL Entry Point Also Gets Called for Thread Attach/Detach

Thread-local variables that are zero-initialized work fine…

```
__declspec(thread) int x; // will be zero-initialized
thread_local       int y; // will be zero-initialized
```

Thread-local variables that can be “statically” initialized work fine…

```
__declspec(thread) int x = 10;
thread_local       int y = 20;
```

or…

```
struct Pair { int a; int b; };

__declspec(thread) Pair x = { 10, 11 };
thread_local       Pair y = { 20, 21 };
```

Thread-Local Storage Works in DLLs. Partially.

Consider thread-local variables that require dynamic initialization:

```
__declspec(thread) unsigned int x = GetCurrentThreadId();
thread_local       unsigned int y = GetCurrentThreadId();
```

These thread-local variables will be correctly initialized for:

- The thread that loaded the DLL
- Any threads that start after the DLL is loaded

These thread-local variables will <u>not</u> be correctly initialized for:

- Any threads that started before the DLL was loaded

For such threads, these thread-local variables will be zero-initialized.

Things Get Tricky for Dynamic Initialization…

Avoiding DLL Hell

“I’m building an app; how do I make sure I load the right DLLs?”

“I’m building a reusable library; how do I make sure I don’t break compatibility?”

Two Sides To This…

The Future / Today (?)

Windows Store Apps / Universal Windows Platform Apps

Desktop Bridge (“Centennial” App Packaging)

Each App has a Well-Defined Package Dependency Graph

This guarantees you full control over the DLLs that you load

Legacy / Supports Downlevel

Install only truly shared components into system directory

Use proper versioned, reference counted setup system like MSI

Also…

If you support plug-ins, define what they can depend on from you and what they need to provide themselves

For Apps

Maintain API/ABI Stability

Don’t expose C++ implementation details across your DLL boundary

Use well-defined ABI concepts for any public, importable APIs (C, COM, WinRT)

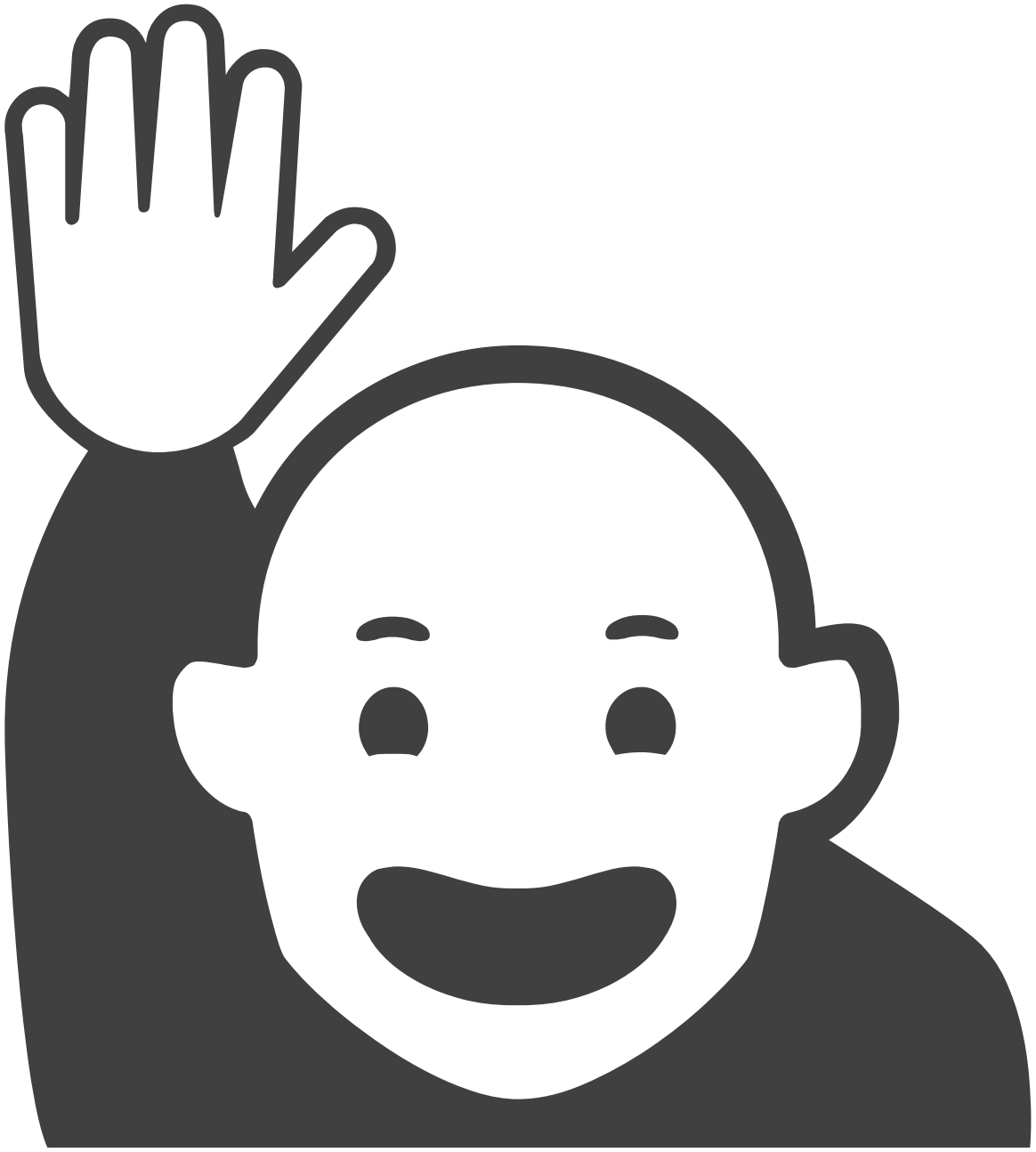
Never remove exports or make breaking behavioral changes to existing exports

If You Make Breaking Changes…

…Rename the DLL

For Shared/Reusable Libraries

The End.



\0