<html><head><style id="style-1-cropbar-clipper">/* Copyright 2014 Evernote Corporation. All rights reserved. */
.en-markup-crop-options {
    top: 18px !important;
    left: 50% !important;
    margin-left: -100px !important;
    width: 200px !important;
    border: 2px rgba(255,255,255,.38) solid !important;
    border-radius: 4px !important;
}

.en-markup-crop-options div div:first-of-type {
    margin-left: 0px !important;
}
</style></head><body><pre style="word-wrap: break-word; white-space: pre-wrap;">Hello M.J.Hsieh,

第一個問題:

```
double pam (int);  // function prototype
int main(int argc, _TCHAR* argv[])
{
        // Step 1: 宣告一個 function pointer
        double (*pf) (int);

        pf = pam;
  ...
}
double pam (int x){
        return  x*2.0;
}
```

why pf and pam has difference address value of pf = 0x004011db
&
pam = 0x00401810 ?

<解答>
==> pf 是一個指標變數內容存放一個指定型態 function 的所在位置,
   這個指標變數佔據記憶體位置為 0x004011db

==> pam 代表 function pam 所在的記憶體位址

Note: 這裡所謂的記憶體位址為 Process 的 Virtual address.

第二個問題:. What is difference from *.lib and *.dll that use DumpBin.exe utility to refer to the export symbol? Because if I edit the *.def file like as:

```
       EXPORTS
         MyFunc
```

in DLL project, it's has difference result about looking up export symbol by *.lib and *.dll
which using use DumpBin.exe ?
<解答>
如果你寫了一堆 functions 且放在不同的 file 中, 你可以把他們全部放到一個 library 檔案中.
這樣可以方便管理以及方便散佈與增加鏈結的效率.

我們都知道要產生一個可執行的檔案你需要兩個階段的工作:

1. Compiler 幫我們建立 Object 檔
2. Linker 進行連結部分,幫我們把 Object 檔參考到的 symbol 連結起來

Linker 接受兩種標準的 COEF 格式的 library 檔[1]
(1) standard libraries: 包含 一堆 object 檔的集合
(2) import libraries: 包含另一個程式所 export 出來的 symbol 資訊
其中 import library 是建立 DLL 檔時, Linker 產生出 import library 來描述 export symbol 的訊息.
上面這兩種格式的附檔名都是 .lib

兩者間的差別在於: standard library 包含了所有的資料, 相對於 import library 則只有包含 export symbol 的link相關資料,完整資料放在另一個檔案 DLL 中.

看到這裡有點累, 別擔心, 最後會有例子把全部串起來!

如果你用靜態鏈結的方式建立你的執行檔, 那麼你就必須指定 .lib 使得 linker 可以在鏈結階段連結你的 library 資訊. 在這裡有個選項

library 資訊. 在這裡有兩個選項:

如果你鏈結的檔案是 standard libraries, 那麼一切都沒事,產生出來的執行檔可以帶到各處執行都沒問題

如果你鏈結的檔案是 import libraries, 那麼你散佈的執行檔只包含部分資訊,你還需要順便散佈 .dll 檔,而且要放在可搜尋到的位置, 這樣執行檔才可以順利執行.

其中, DLL 檔就是所謂的 Dynamic link libraries 的縮寫. 使用 DLL + import library 可以讓你的執行檔只包含必要的部分, 其他共用的部分可以放在 DLL 檔讓其他程式共享. 這使得你的執行檔可以佔比較小的空間, 而且若 DLL 檔已經被其他 process 載入時,你的執行檔就不用載入該共享的 DLL 檔而使用 map 的方式, 將 DLL 檔的 code map 到你process 的 virtual space 中, 進而節省 Loading 的時間使得開啟你的程式. 會有比較快的反應時間.

更進一步的來說,你還可以使用 LoadLibarary 指令,讓你要用到某個 .DLL export 的symbol的時候, 才 map 該 DLL , 不用的時候使用 FreeLibary 指令 unmap DLL, 使你更精確的控制你 process 所使用的空間. 順便提一下在 linux(或 UNIX) 世界也有 DLL, 只是他的附檔名通常被稱為 .so.

那麼要如何建立 DLL 與使用 DLL ?

基本上, 我們可以在 source code 加上下面的敘述將一個程式的 symbol export 出去[2]:

```
__declspec(dllexport)
```

如果你使用 implicit link 使用你的 DLL, 那麼你的程式很簡單的可以直接使用 DLL export 出來的symbol . 例如 直接呼叫 DLL export 出來的 function Add.

但是在這裡有一個問題, 那就是你的source code 中直接使用 Add 這個 symbol, C++ 會告訴我們

```
-----------------------------------------------
 error LNK2019: unresolved external symbol 的錯誤
-----------------------------------------------
```

這是因為 linker 不知道 你使用的 Add 是放在其他的程式檔(某個DLL)中, 所以才會產生這個錯誤.
在這裡,你應該在 project 中或 source code 中指出"Add"這個 symbol 的詳細資訊, 才會正確連結成功.
而這個資訊就放在 .lib 檔中.
在 source code 指定 .lib 的範例

```
-------------------------------------------------------------------------------
            #pragma comment( lib, "C:\\Documents and Settings\\jing\\桌面
\\MyDLL\\DEBUG\\MyDLL.lib" )
-------------------------------------------------------------------------------
```

結合上面的說法:
我們寫個範例給大家看看:

Step 1: 先用 整合式開發環境(VC) 建立一個 DLL 專案

Step 2: 寫 header file

```
--------------MyLib.h---------------
// 宣告要 export 的 function 以及全區域變數
// 這個檔案要include任何想要使用 DLL 中function 的 executable file 中
#ifdef MYLIBExport
   #define  MYLIBAPI extern "C" __declspec(dllexport)   // 說明我要 export
#else
   #define  MYLIBAPI extern "C" __declspec(dllimport)   // 說明我要 import
#endif

MYLIBAPI int Add(int nLeft, int nRight);
----------------------------------------
```

Step 3: 寫 cpp 檔

```
------------- MyLib.cpp---------------------
#include &lt;windows.h&gt;

#define MYLIBExport            // 表示這個 .cpp 檔要 export symbol
#include "MyLib.h"
MYLIBAPI int Add(int nLeft, int nRight) {
   g_nResult = nLeft + nRight;
   return(g_nResult);
}
----------------------------------------
```

Step 4: 編譯與鏈結: VC 就會在 的專案目錄下的 Debug 建立 .lib 檔與 .dll 檔

使用 DLL 檔

如果你使用的是 implicit link 你的 DLL, 那麼前面說過,你需要 .lib 幫忙產生執行檔.

範例如下:

Step 1: 利用 VC 建立新的專案: UseDLL

Step 2: 指定 DLL 的 .h 檔位置,把他 include 進來.

```
#include "C:\\Documents and Settings\\jing\\桌面\\MyDLL\\MyLib.h"
```

Step 3: 加入 lib 檔到你的 source code

```
#pragma comment( lib, "C:\\Documents and Settings\\jing\\桌面
\\MyDLL\\DEBUG\\MyDLL.lib" )
```

Step 4: 編譯執行(大功告成)

所以你的程式應該長的像這樣:

```
-------------------------------------
#include "C:\\Documents and Settings\\jing\\桌面\\MyDLL\\MyLib.h"
#pragma comment( lib, "C:\\Documents and Settings\\jing\\桌面\\MyDLL\\DEBUG\\MyDLL.lib" )

int _tmain(int argc, _TCHAR* argv[]){
     int c=Add(1,2);                 // &lt;&lt; -- 直接使用 export 的function
     return 0;
}
-------------------------------------
```

若你使用的是 explicit link 方式使用你的 DLL, 那麼你的 source 就不能直接使用 DLL export 出來的 symbol了, 情況是你必須寫程式自己載入 DLL 檔,接著利用取出你要的存取的 fucntion 所在的位址, 然後使用 function pointer 間接的呼叫 function.

所以情況如下:

```
          ---------------------------------------------
          typedef int (*MYPROC)(int,int);
          void main(){
              HINSTANCE hinstDll  = LoadLibrary("MyDLL"); // Step 1: 將 DLL 載入
              MYPROC ProcAdd;
              ProcAdd=(MYPROC)GetProcAddress(hinstDll,"Add"); // Step 2: 宣告指標
              int data=(ProcAdd)(2,4);                        // Step 3: 呼叫
function
          }
          ---------------------------------------------
```

你可以看到, 在這裡完全沒有直接使用 DLL export 出來的 symbol, 我們是利用 Win32 API 幫我們間接取得 呼叫 function 的 map 位址,然後間接的呼叫. 所以在 linking 階段不需要使用 lib 檔提供有關 symbol "Add" 的資訊.

所以結論是: .lib 檔描述的是 提供 linker 有關 export symbol 的鏈結資訊.

.dll 檔內容是包含 function 的可 relocated object code,

在其中有一個 export section 紀錄著這個 DLL 所有 exported 的 symbol.

這兩個檔案可能用不同的方式是描述 exported symbol.

你或許會問: 為什麼要分成這兩個檔案呢? 在 implicit link 時就直接使用 .dll 就好了嘛,

.dll 就有 export section 了,直接擷取該節區就有資料何必使用 .lib ?

我的解答是:

透過 .lib 間接描述 .dll export symbol 有好處. 因為一旦 linking 階段完成靜態鏈結你的source code 與 .lib symbol 也就完成了, 所以當 DLL 檔因為新版本出現需要改變時,只要直接更換 DLL 檔就可以了不需要換重新編譯執行檔.

[1] <ms-help://MS.MSDNQTR.2004JAN.1033/vccore/html/_core_..LIB_Files_as_Linker_Input.htm>

[2] ms-help://MS.MSDNQTR.2004JAN.1033/vclang/html/_pluslang_The_dllexport_and_dllimport_Attributes.htm

.def 檔簡介

Module-definition (.def) 檔案提供有關要被連結程式的 exports symbol, attributes 等相關資訊給 linker. 要注意的是: 在 .def 檔中 是大小寫有分別的. EXPORTS statement 引導出一個程式段落所

export 出來的 funciton 或 data. 這些 function 或 data 被稱作 definitions. 他的格式如下:

```
---------------
 EXPORTS
 definitions
---------------
```

每一個 definitions 必須以行為單位, 意思是如果你要定義新的 definition, 請你寫在下一行.
而 EXPORTS 可以與 defineition 同一行. 整個完整的 definitions 格式如下:

```
        entryname[=internalname] [@ordinal [NONAME]] [PRIVATE] [DATA]
```

-entryname: 你要 export 的 function 或變數名稱

</pre></body></html>