Dynamic-Link Library (DLL)

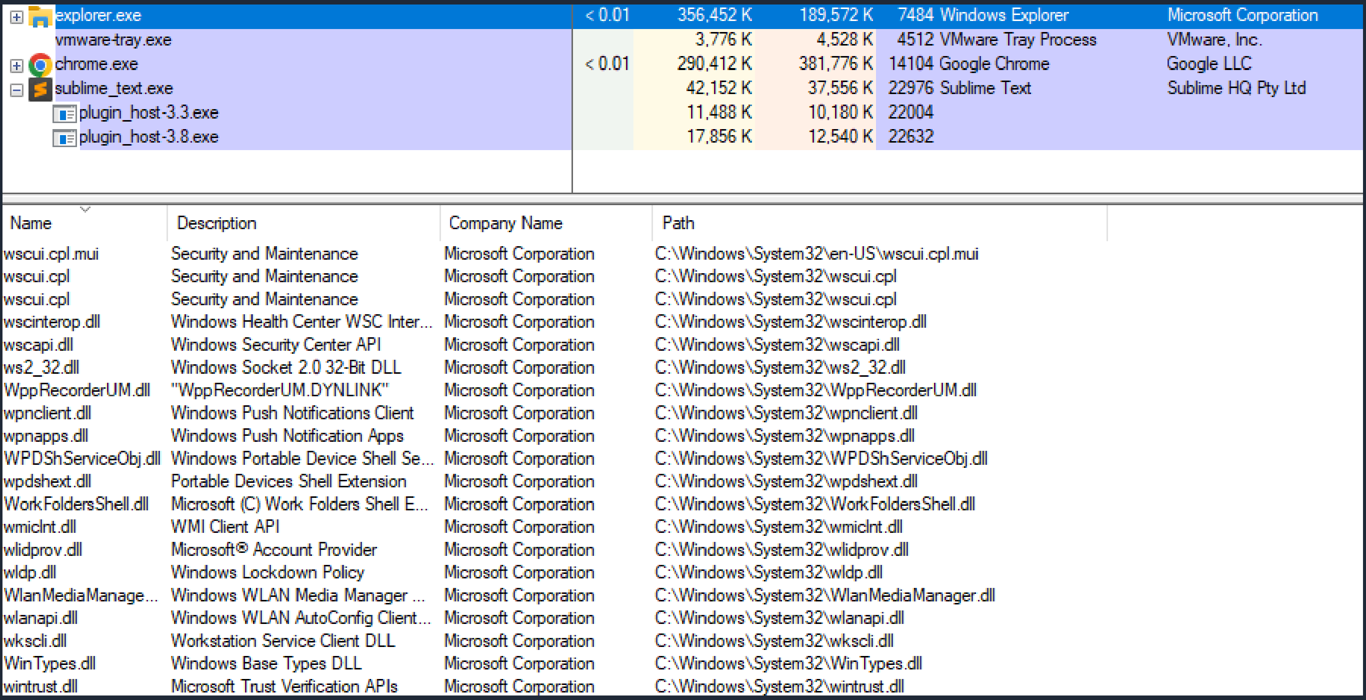
Introduction

Both `.exe` and `.dll` file types are considered portable executable formats but there are differences between the two. This module explains the difference between the two file types.

What is a DLL?

DLLs are shared libraries of executable functions or data that can be used by multiple applications simultaneously. They are used to export functions to be used by a process. Unlike EXE files, DLL files cannot execute code on their own. Instead, DLL libraries need to be invoked by other programs to execute the code. As previously mentioned, the `CreateFileW` is exported from `kernel32.dll`, therefore if a process wants to call that function it would first need to load `kernel32.dll` into its address space.

Some DLLs are automatically loaded into every process by default since these DLLs export functions that are necessary for the process to execute properly. A few examples of these DLLs are `ntdll.dll`, `kernel32.dll` and `kernelbase.dll`. The image below shows several DLLs that are currently loaded by the `explorer.exe` process.

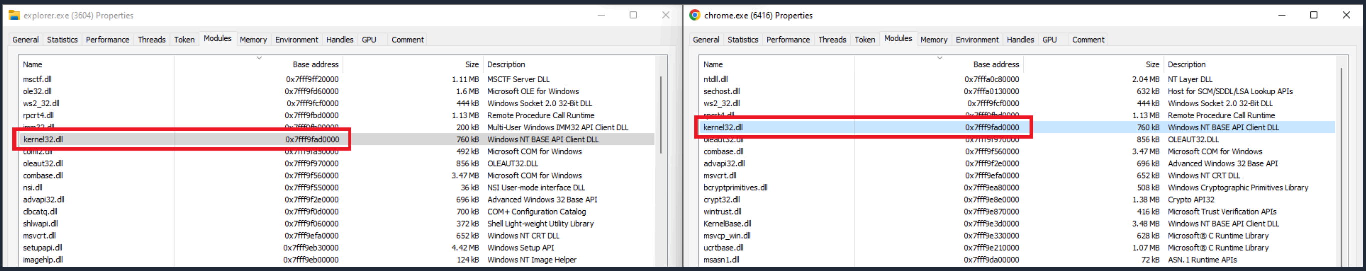


explorer.exe	< 0.01	356,452 K	189,572 K	7484	Windows Explorer	Microsoft Corporation
vmware-tray.exe		3,776 K	4,528 K	4512	VMware Tray Process	VMware, Inc.
chrome.exe	< 0.01	290,412 K	381,776 K	14104	Google Chrome	Google LLC
sublime_text.exe		42,152 K	37,556 K	22976	Sublime Text	Sublime HQ Pty Ltd
plugin_host-3.3.exe		11,488 K	10,180 K	22004		
plugin_host-3.8.exe		17,856 K	12,540 K	22632		

Name	Description	Company Name	Path
wscui.cpl.mui	Security and Maintenance	Microsoft Corporation	C:\Windows\System32\en-US\wscui.cpl.mui
wscui.cpl	Security and Maintenance	Microsoft Corporation	C:\Windows\System32\wscui.cpl
wscui.cpl	Security and Maintenance	Microsoft Corporation	C:\Windows\System32\wscui.cpl
wscinterop.dll	Windows Health Center WSC Inter...	Microsoft Corporation	C:\Windows\System32\wscinterop.dll
wscapi.dll	Windows Security Center API	Microsoft Corporation	C:\Windows\System32\wscapi.dll
ws2_32.dll	Windows Socket 2.0 32-Bit DLL	Microsoft Corporation	C:\Windows\System32\ws2_32.dll
WppRecorderUM.dll	"WppRecorderUM.DYNLINK"	Microsoft Corporation	C:\Windows\System32\WppRecorderUM.dll
wpnclient.dll	Windows Push Notifications Client	Microsoft Corporation	C:\Windows\System32\wpnclient.dll
wpnapps.dll	Windows Push Notification Apps	Microsoft Corporation	C:\Windows\System32\wpnapps.dll
WPDShServiceObj.dll	Windows Portable Device Shell Se...	Microsoft Corporation	C:\Windows\System32\WPDShServiceObj.dll
wpdshext.dll	Portable Devices Shell Extension	Microsoft Corporation	C:\Windows\System32\wpdshext.dll
WorkFoldersShell.dll	Microsoft (C) Work Folders Shell E...	Microsoft Corporation	C:\Windows\System32\WorkFoldersShell.dll
wmiclnt.dll	WMI Client API	Microsoft Corporation	C:\Windows\System32\wmiclnt.dll
wlidprov.dll	Microsoft® Account Provider	Microsoft Corporation	C:\Windows\System32\wlidprov.dll
wldp.dll	Windows Lockdown Policy	Microsoft Corporation	C:\Windows\System32\wldp.dll
WlanMediaManage...	Windows WLAN Media Manager ...	Microsoft Corporation	C:\Windows\System32\WlanMediaManager.dll
wlanapi.dll	Windows WLAN AutoConfig Client...	Microsoft Corporation	C:\Windows\System32\wlanapi.dll
wkscli.dll	Workstation Service Client DLL	Microsoft Corporation	C:\Windows\System32\wkscli.dll
WinTypes.dll	Windows Base Types DLL	Microsoft Corporation	C:\Windows\System32\WinTypes.dll
wintrust.dll	Microsoft Trust Verification APIs	Microsoft Corporation	C:\Windows\System32\wintrust.dll

System-Wide DLL Base Address

The Windows OS uses a system-wide DLL base address to load some DLLs at the same base address in the virtual address space of all processes on a given machine to optimize memory usage and improve system performance. The following image shows `kernel32.dll` being loaded at the same address (`0x7fff9fad0000`) among multiple running processes.



explorer.exe (3604) Properties

Name	Base address	Size	Description
msctf.dll	0x7fff9ff20000	1.11 MB	MSCTF Server DLL
ole32.dll	0x7fff9fd60000	1.6 MB	Microsoft OLE for Windows
ws2_32.dll	0x7fff9fcf0000	444 kB	Windows Socket 2.0 32-Bit DLL
rpcrt4.dll	0x7fff9fbd0000	1.13 MB	Remote Procedure Call Runtime
imm32.dll	[illegible]	200 kB	Multi-User Windows IMM32 API Client DLL
kernel32.dll	0x7fff9fad0000	760 kB	Windows NT BASE API Client DLL
coml2.dll	0x7fff9fa50000	492 kB	Microsoft COM for Windows
oleaut32.dll	0x7fff9f970000	856 kB	OLEAUT32.DLL
combase.dll	0x7fff9f560000	3.47 MB	Microsoft COM for Windows
nsi.dll	0x7fff9f550000	36 kB	NSI User-mode interface DLL
advapi32.dll	0x7fff9f2e0000	696 kB	Advanced Windows 32 Base API
clbcatq.dll	0x7fff9f0d0000	700 kB	COM+ Configuration Catalog
shlwapi.dll	0x7fff9f060000	372 kB	Shell Light-weight Utility Library
msvcrt.dll	0x7fff9efa0000	652 kB	Windows NT CRT DLL
setupapi.dll	0x7fff9eb30000	4.42 MB	Windows Setup API
imagehlp.dll	0x7fff9eb00000	124 kB	Windows NT Image Helper

chrome.exe (6416) Properties

Name	Base address	Size	Description
ntdll.dll	0x7fffa0c80000	2.04 MB	NT Layer DLL
sechost.dll	0x7fffa0130000	632 kB	Host for SCM/SDDL/LSA Lookup APIs
ws2_32.dll	0x7fff9fcf0000	444 kB	Windows Socket 2.0 32-Bit DLL
rpcrt4.dll	0x7fff9fbd0000	1.13 MB	Remote Procedure Call Runtime
kernel32.dll	0x7fff9fad0000	760 kB	Windows NT BASE API Client DLL
oleaut32.dll	0x7fff9f970000	856 kB	OLEAUT32.DLL
combase.dll	0x7fff9f560000	3.47 MB	Microsoft COM for Windows
advapi32.dll	0x7fff9f2e0000	696 kB	Advanced Windows 32 Base API
msvcrt.dll	0x7fff9efa0000	652 kB	Windows NT CRT DLL
bcryptprimitives.dll	0x7fff9ea80000	508 kB	Windows Cryptographic Primitives Library
crypt32.dll	0x7fff9e8e0000	1.38 MB	Crypto API32
wintrust.dll	0x7fff9e870000	416 kB	Microsoft Trust Verification APIs
KernelBase.dll	0x7fff9e3d0000	3.48 MB	Windows NT BASE API Client DLL
msvcp_win.dll	0x7fff9e330000	628 kB	Microsoft® C Runtime Library
ucrtbase.dll	0x7fff9e210000	1.07 MB	Microsoft® C Runtime Library
msasn1.dll	0x7fff9da00000	72 kB	ASN.1 Runtime APIs

Why Use DLLs?

There are several reasons why DLLs are very often used in Windows:

1. **Modularization of Code** - Instead of having one massive executable that contains the entire functionality, the code is divided into several independent libraries with each library being focused on specific functionality. Modularization makes it easier for developers during development and debugging.
2. **Code Reuse** - DLLs promote code reuse since a library can be invoked by multiple processes.
3. **Efficient Memory Usage** - When several processes need the same DLL, they can save memory by sharing that DLL instead of loading it into the process's memory.

DLL Entry Point

DLLs can optionally specify an entry point function that executes code when a certain task occurs such as when a process loads the DLL library. There are 4 possibilities for the entry point being called:

- `DLL_PROCESS_ATTACHED` - A process is loading the DLL.
- `DLL_THREAD_ATTACHED` - A process is creating a new thread.
- `DLL_THREAD_DETACH` - A thread exits normally.
- `DLL_PROCESS_DETACH` - A process unloads the DLL.

Sample DLL Code

The code below shows a typical DLL code structure.

```
BOOL APIENTRY DllMain(
    HANDLE hModule,             // Handle to DLL module
    DWORD ul_reason_for_call,   // Reason for calling function
    LPVOID lpReserved           // Reserved
) {

    switch (ul_reason_for_call) {
        case DLL_PROCESS_ATTACHED: // A process is loading the DLL.
        // Do something here
        break;
        case DLL_THREAD_ATTACHED: // A process is creating a new thread.
        // Do something here
        break;
        case DLL_THREAD_DETACH: // A thread exits normally.
        // Do something here
        break;
        case DLL_PROCESS_DETACH: // A process unloads the DLL.
        // Do something here
        break;
    }
    return TRUE;
}
```

Exporting a Function

DLLs can export functions that can then be used by the calling application or process. To export a function it needs to be defined using the keywords `extern` and `__declspec(dllexport)`. An example exported function `HelloWorld` is shown below.

```
////// sampleDLL.dll //////

extern __declspec(dllexport) void HelloWorld(){
// Function code here
}
```

Dynamic Linking

It's possible to use the `LoadLibrary`, `GetModuleHandle` and `GetProcAddress` WinAPIs to import a function from a DLL. This is referred to as dynamic linking. This is a method of loading and linking code (DLLs) at runtime rather than linking them at compile time using the linker and import address table.

There are several advantages of using dynamic linking, these are documented by Microsoft here.

This section walks through the steps of loading a DLL, retrieving the DLL's handle, retrieving the exported function's address and then invoking the function.

Loading a DLL

Calling a function such as MessageBoxA in an application will force the Windows OS to load the DLL exporting the `MessageBoxA` function into the calling process's memory address space, which in this case is `user32.dll`. Loading `user32.dll` was done automatically by the OS when the process started and not by the code.

However, in some cases such as the `HelloWorld` function in `sampleDLL.dll`, the DLL may not be loaded into memory. For the application to call the `HelloWorld` function, it first needs to retrieve the DLL's handle that is exporting the function. If the application doesn't have `sampleDLL.dll` loaded into memory, it would require the usage of the LoadLibrary WinAPI, as shown below.

```
HMODULE hModule = LoadLibraryA("sampleDLL.dll"); // hModule now contain sampleDLL.dll's handle
```

Retrieving a DLL's Handle

If `sampleDLL.dll` is already loaded into the application's memory, one can retrieve its handle via the GetModuleHandle WinAPI function without leveraging the `LoadLibrary` function.

```
HMODULE hModule = GetModuleHandleA("sampleDLL.dll");
```

Retrieving a Function's Address

Once the DLL is loaded into memory and the handle is retrieved, the next step is to retrieve the function's address. This is done using the GetProcAddress WinAPI which takes the handle of the DLL that exports the function and the function name.

```
PVOID pHelloWorld = GetProcAddress(hModule, "HelloWorld");
```

Invoking The Function

Once `HelloWorld`'s address is saved into the `pHelloWorld` variable, the next step is to perform a type-cast on this address to `HelloWorld`'s function pointer. This function pointer is required in order to invoke the function.

```
// Constructing a new data type that represents HelloWorld's function
pointer
typedef void (WINAPI* HelloWorldFunctionPointer)();

void call(){
    HMODULE hModule = LoadLibraryA("sampleDLL.dll");
    PVOID pHelloWorld = GetProcAddress(hModule, "HelloWorld");
    // Type-casting the 'pHelloWorld' variable to be of type
'HelloWorldFunctionPointer'
    HelloWorldFunctionPointer HelloWorld =
(HelloWorldFunctionPointer)pHelloWorld;
    HelloWorld();   // Calling the 'HelloWorld' function via its
function pointer
}
```

Dynamic Linking Example

The code below demonstrates another simple example of dynamic linking where `MessageBoxA` is called. The code assumes that `user32.dll`, the DLL that exports that function, isn't loaded into memory. Recall that if a DLL isn't loaded into memory the usage of `LoadLibrary` is required to load that DLL into the process's address space.

```
typedef int (WINAPI* MessageBoxAFunctionPointer)( // Constructing a
new data type, that will represent MessageBoxA's function pointer
  HWND          hWnd,
  LPCSTR        lpText,
  LPCSTR        lpCaption,
  UINT          uType
);

void call(){
    // Retrieving MessageBox's address, and saving it to
'pMessageBoxA' (MessageBoxA's function pointer)
    MessageBoxAFunctionPointer pMessageBoxA =
(MessageBoxAFunctionPointer)GetProcAddress(LoadLibraryA("user32.dll")
, "MessageBoxA");
    if (pMessageBoxA != NULL){
        // Calling MessageBox via its function pointer if not null
        pMessageBoxA(NULL, "MessageBox's Text", "MessageBox's
Caption", MB_OK);
    }
}
```

Function Pointers

For the remainder of the course, the function pointer data types will have a naming convention that uses the WinAPI's name prefixed with `fn`, which stands for "function pointer". For example, the above `MessageBoxAFunctionPointer` data type will be represented as `fnMessageBoxA`. This is used to maintain simplicity and improve clarity throughout the course.

Rundll32.exe

There are a couple of ways to run exported functions without using a programmatical method. One common technique is to use the rundll32.exe binary. `Rundll32.exe` is a built-in Windows binary that is used to run an exported function of a DLL file. To run an exported function use the following command:

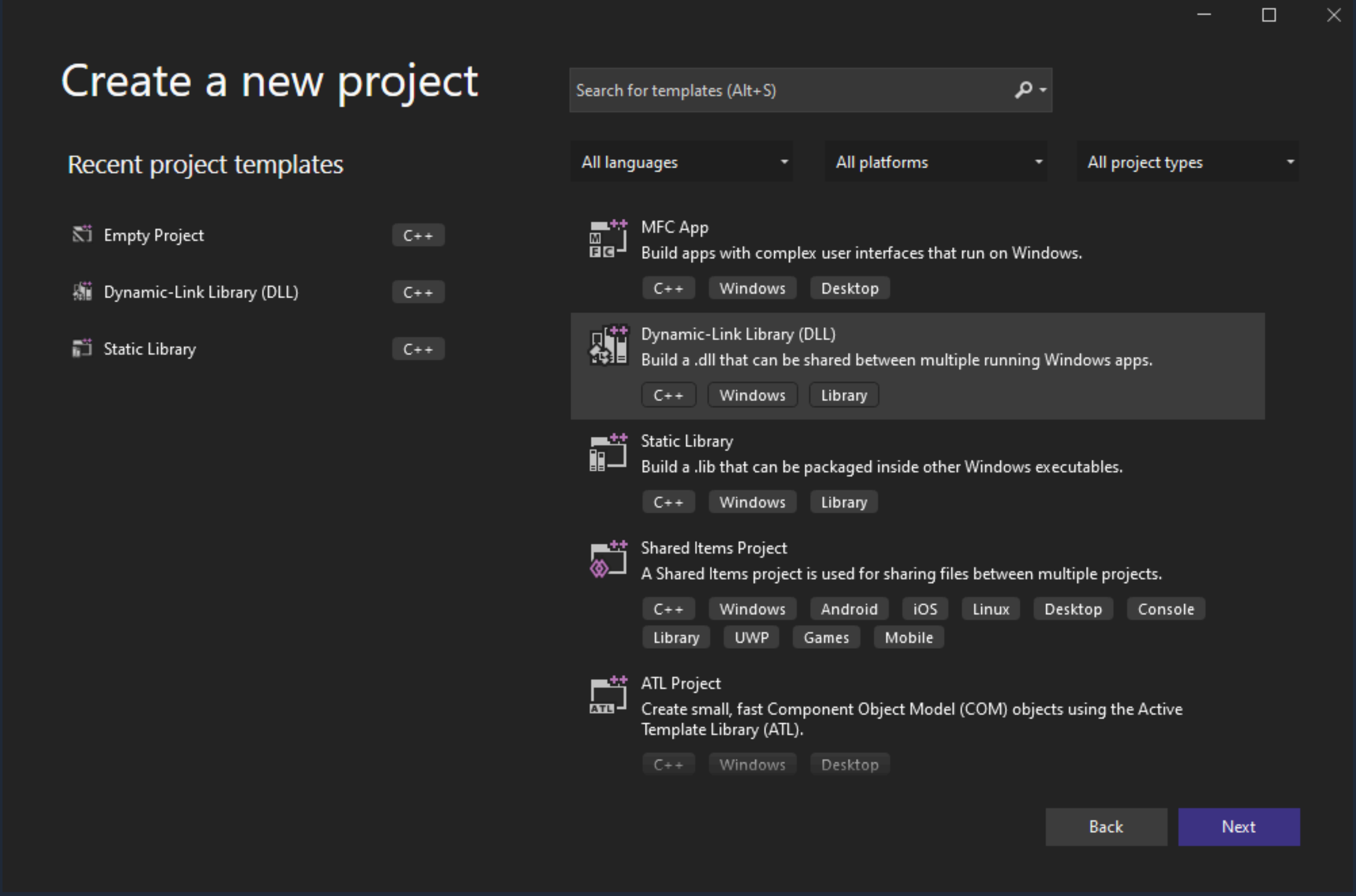
```
rundll32.exe <dllname>, <function exported to run>
```

For example, `User32.dll` exports the function `LockWorkStation` which locks the machine. To run the function, use the following command:

```
rundll32.exe user32.dll,LockWorkStation
```

Creating a DLL File With Visual Studio

To create a DLL file, launch Visual studio and create a new project. When given the project templates, select the `Dynamic-Link Library (DLL)` option.



Create a new project

Recent project templates

- Empty Project C++
- Dynamic-Link Library (DLL) C++
- Static Library C++

MFC App
Build apps with complex user interfaces that run on Windows.
C++ Windows Desktop

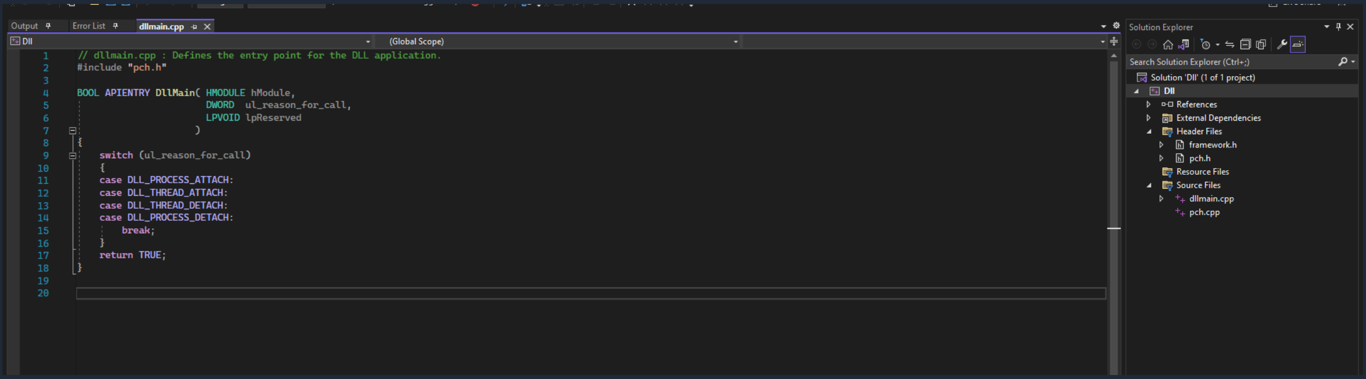
Dynamic-Link Library (DLL)
Build a .dll that can be shared between multiple running Windows apps.
C++ Windows Library

Static Library
Build a .lib that can be packaged inside other Windows executables.
C++ Windows Library

Shared Items Project
A Shared Items project is used for sharing files between multiple projects.
C++ Windows Android iOS Linux Desktop Console Library UWP Games Mobile

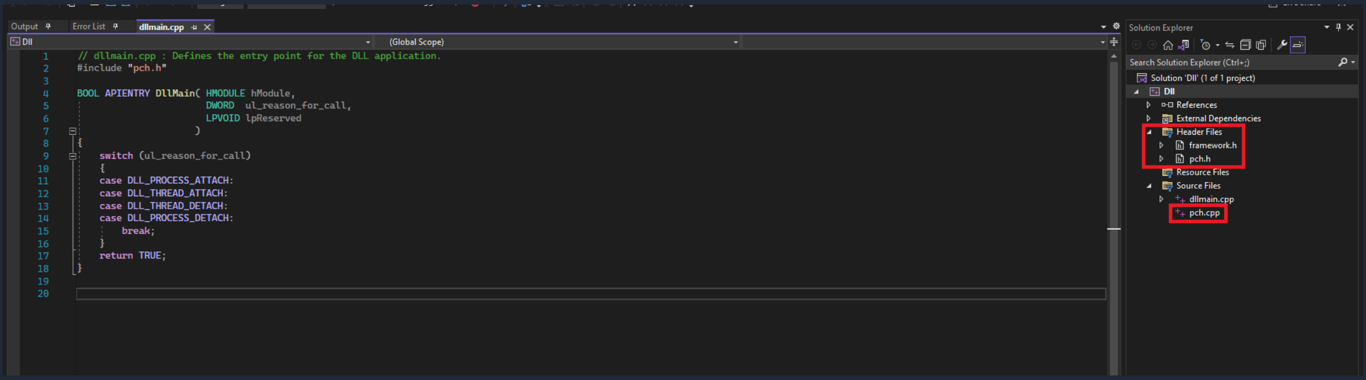
ATL Project
Create small, fast Component Object Model (COM) objects using the Active Template Library (ATL).
C++ Windows Desktop

Next, select the location where to save the project files. When that's done, the following C code should appear.

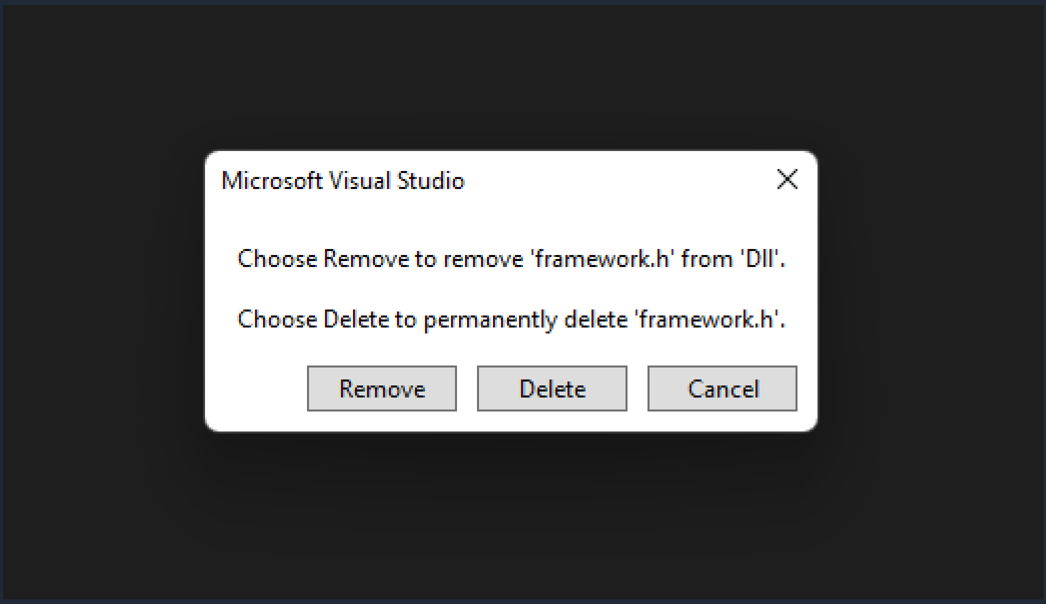


```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

The provided DLL template comes with `framework.h`, `pch.h` and `pch.cpp` which are known as Precompiled Headers. These are files used to make the project compilation faster for large projects. It is unlikely that these will be required in this situation and therefore it is recommended to delete these files. To do so, highlight the file and press the delete key and select the 'Delete' option.



```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

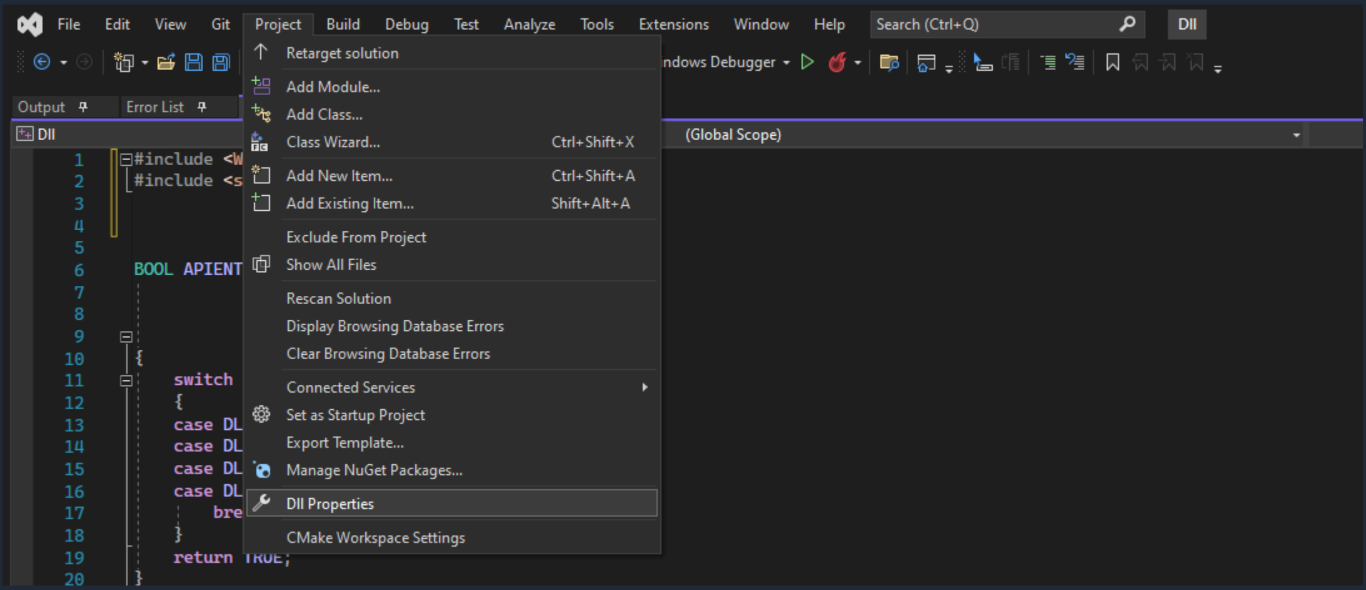


Microsoft Visual Studio

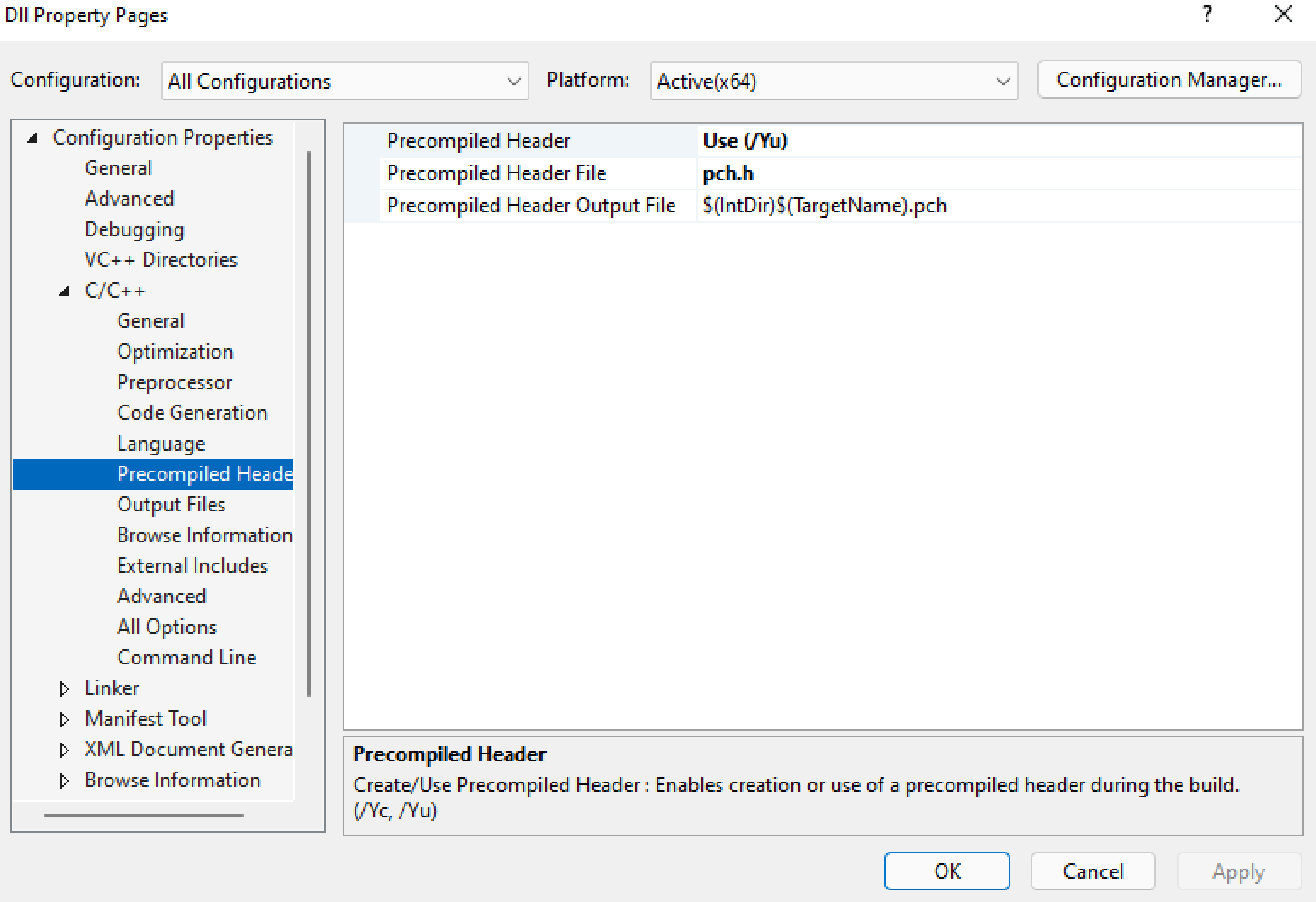
Choose Remove to remove 'framework.h' from 'Dll'.

Choose Delete to permanently delete 'framework.h'.

After deleting the precompiled headers, the compiler's default settings must be changed to confirm that precompiled headers should not be used in the project.



Go to **C/C++ > Advanced Tab**



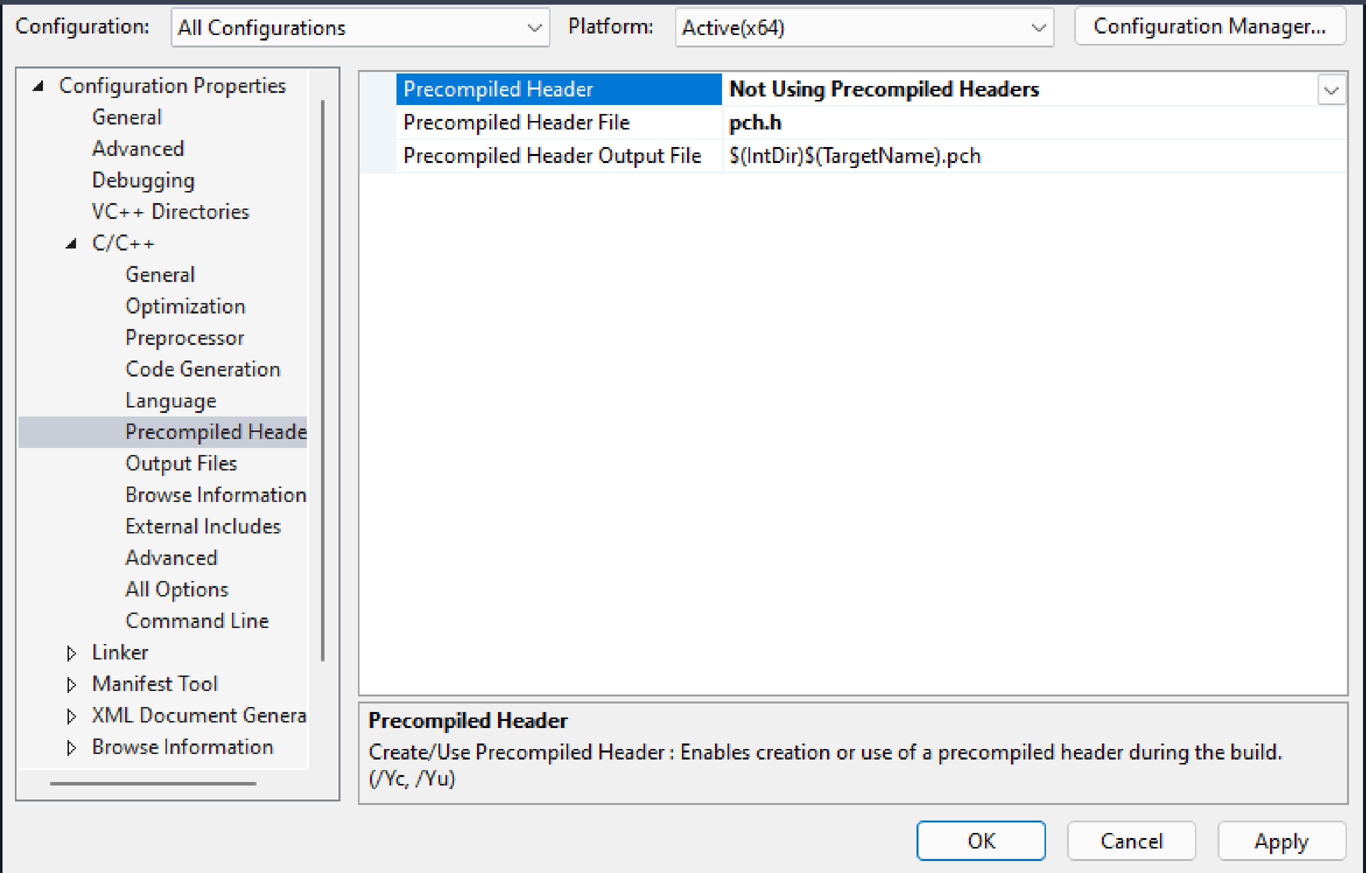
Configuration: All Configurations   Platform: Active(x64)

Precompiled Header	**Use (/Yu)**
Precompiled Header File	**pch.h**
Precompiled Header Output File	$(IntDir)$(TargetName).pch

Precompiled Header
Create/Use Precompiled Header : Enables creation or use of a precompiled header during the build. (/Yc, /Yu)

Change the 'Precompiled Header' option to 'Not Using Precompiled Headers' and press

Previous Modules Undo Next



Configuration: All Configurations   Platform: Active(x64)

Precompiled Header	**Not Using Precompiled Headers**
Precompiled Header File	**pch.h**
Precompiled Header Output File	$(IntDir)$(TargetName).pch

Precompiled Header
Create/Use Precompiled Header : Enables creation or use of a precompiled header during the build. (/Yc, /Yu)

Finally, change the `dllmain.cpp` file to `dllmain.c`. This is required since the provided code snippets in Maldev Academy use C instead of C++. To compile the program, click Build > Build Solution and a DLL will be created under the *Release* or *Debug* folder, depending on the compile configuration.