If you're going to configure a header file, you have to do it before you include the header file

devblogs.microsoft.com/oldnewthing/20211209-00

December 9, 2021

Raymond Chen

In Windows, if you want the `windows.h` header file to default to the Unicode character set, you need to define the `UNICODE` symbol:

```
#define UNICODE
#include <windows.h>
```

There's no point defining the symbol after the horse has left the barn:

```
// Code in italics is wrong
#include <windows.h>
#define UNICODE
```

Related: Don't forget to #define UNICODE if you want Unicode.

Now, sure, a mistake like that is pretty obvious and easy to spot.

But other mistakes might not be so easy to find.

```
#include <stdio.h>
#include <contoso.h>
#include <fastmalloc.h>
#define UNICODE
#include <windows.h>

extern HWND g_mainWindow;

void UpdateTitle(PCWSTR title)
{
  SetWindowText(g_mainWindow, title);
}
```

This looks perfectly fine: We defined the `UNICODE` symbol before we included `windows.h`, but we still get compiler errors that suggest that the symbol didn't take effect:

```
error C2664: 'BOOL SetWindowTextA(HWND,const char *)': cannot convert argument 2 from
'const wchar_t *' to 'const char *'
note: Types pointed to are unrelated; conversion requires reinterpret_cast, C-style
cast or function-style cast
```

The error message tells us that the compiler thinks that `SetWindowText` redirects to `SetWindowTextA`, and that function expects an ANSI string, not a Unicode one. But how can that be? We did the right thing and defined `UNICODE` symbol immediately before including `windows.h`.

Yes, you defined it immediately before you included `windows.h`. But your inclusion of `windows.h` isn't the one that counted.

Somewhere in the chain of `#include` files that came before you included `windows.h` is somebody that did their own `#include <windows.h>`. That first inclusion did not have the `UNICODE` symbol defined, so the result was that all of the Windows macros redirected to the ANSI versions. You reconfigured the `windows.h` header afterward, but by then it was too late.

You'll need to move the definition to somewhere that occurs before the point at which the `windows.h` header is included for the first time. The safest place is to do it before including *anything*.

```
#define UNICODE
#include <stdio.h>
#include <contoso.h>
#include <fastmalloc.h>
#include <windows.h>
```

Bonus chatter: If you want to find out who is doing the early inclusion of `windows.h`, you can pass the `/showIncludes` command line option to the Microsoft Visual C++ compiler. This can be configured in Visual Studio under *Project → Configuration Properties → C/C++ → Advanced → Show Includes = Yes (/showIncludes)*. This will generate a hierarchy tree of every included file, and you can search it to see who included `windows.h`.

For gcc, you don't even need to do that much. You can reuse a trick we saw a little while ago: Give a macro a conflicting definition and wait for the fireworks.

```
#define CreateFile who_includes_windows_h_first
#define UNICODE
#include <stdio.h>
#include <contoso.h>
#include <fastmalloc.h>
#include <windows.h>
```

You will then get told

```
in file included from /sdk/fileapifromapp.h:19:
                 from /sdk/winbase.h:42:
                 from /sdk/windows.h:171:
                 from /contoso/inc/internal/config.h:84:
                 from /contoso/inc/contoso.h:20:
/sdk/fileapi.h:84: warning: "CreateFile" redefined
```

Bingo, it's `contoso.h`, via its helper include file `internal/config.h`.

Raymond Chen

Follow