# Why do member functions need to be "static" to be used as a callback?

devblogs.microsoft.com/oldnewthing/20040109-00

January 9, 2004

Raymond Chen

As we learned yesterday, nonstatic member functions take a secret “this” parameter, which makes them incompatible with the function signature required by Win32 callbacks. Fortunately, nearly all callbacks provide some way of providing context. You can shove the “this” pointer into the context so you can reconstruct the source object. Here’s an example:

```
class SomeClass {
 ...
 static DWORD CALLBACK s_ThreadProc(LPVOID lpParameter)
 {
  return ((SomeClass*)lpParameter)->ThreadProc();
 }
 DWORD ThreadProc()
 {
  ... fun stuff ...
 }
};
```

Some callback function signatures place the context parameter (also known as “reference data”) as the first parameter. How convenient, for the secret “this” parameter is also the first parameter. Looking at the various calling conventions available to us, it sure looks like the `__stdcall` calling convention for **member functions** matches our desired stack layout rather well. Let’s take `WAITORTIMERCALLBACK` for example:

| __stdcall callback | | __stdcall method call | | thiscall method call | |
|---|---|---|---|---|---|
| .. rest of stack .. | | .. rest of stack .. | | .. rest of stack .. | |
| TimerOrWaitFired | | TimerOrWaitFired | | TimerOrWaitFired | <- ESP |
| lpParameter | <- ESP | this | <- ESP | | |

Well, "thiscall" doesn't match, but the two "__stdcall"s do. Fortunately the compiler is smart enough to recognize this and can optimize the `s_ThreadProc` static method to nothing if you just give it enough of a nudge:

```
class SomeClass {
 ...
 static DWORD CALLBACK s_ThreadProc(LPVOID lpParameter)
 {
  return ((SomeClass*)lpParameter)->ThreadProc();
 }
 DWORD __stdcall ThreadProc()
 {
  ... fun stuff ...
 }
};
```

If you look at the code generation for the `s_ThreadProc` function, you'll see that has been reduced to nothing but a jump instruction, since the compiler has realized that the two calling conventions coincide here so there is no actual translation to do.

```
?s_ThreadProc@SomeClass@@SGKPAX@Z PROC NEAR
  jmp     ?ThreadProc@SomeClass@@QAGKXZ
?s_ThreadProc@SomeClass@@SGKPAX@Z ENDP
```

Now some people would take this one step further and just cast the second parameter to `CreateThread` to `LPTHREAD_START_ROUTINE` and get rid of the helper `s_ThreadProc` function entirely. **I strongly advise against this.** I have seen too many people cause trouble by miscasting function pointers; more on this in a future entry.

Although we took advantage above of a coincidence between the two `__stdcall` calling conventions, we did not **rely** on it. If the coincidence in calling conventions fails to occur, the code is still correct. This is important when it comes time to port this code to another architecture, one where the coincidence may longer be true!

Raymond Chen

**Follow**