Wrapper templates to make writing callback functions slightly easier

devblogs.microsoft.com/oldnewthing/20040719-00

July 19, 2004

Raymond Chen

I previously discussed why callback functions must be static if they are member functions.

Writing the correct prototype for the callback function is usually somewhat clumsy. It's not hard. Just clumsy.

```
class Sample {
 static DWORD CALLBACK StaticThreadProc(LPVOID *lpParameter)
 {
  Sample *self = reinterpret_cast<Sample*>(lpParameter);
  return self->RealThreadProc();
 }
 DWORD __stdcall RealThreadProc()
 {
   ... do stuff ...
 }
 void DoSomething()
 {
   ... CreateThread(NULL, 0, StaticThreadProc, this, 0, &dwTid); ...
 }
};
```

(If you read my previous article, you'd recognizing sticking a __stdcall in the declaration for RealThreadProc as a micro-optimization.)

Every class that has a thread procedure needs this "trampoline" function `StaticThreadProc` that has the correct function signature, then massages it into something that is easier to work with (in this case, an object pointer and method call). Well, okay, you could do the work directly in the trampoline if you wanted to, but it's usually much more convenient to put the bulk of the work in a proper member function so you have access to all the "this" shorthand.

If you do this a lot, you can write a template function to do the boring grunt work, freeing your brain to do "real thinking".

```
template<class T, DWORD (__stdcall T::*F)()>
DWORD CALLBACK ThreadProc(void *p)
{
    return ((reinterpret_cast<T*>(p))->*F)();
}
```

This template function declares a templatized thread procedure. Notice that the calling convention for the `ThreadProc` template function is correct for a thread function, so this guy can be passed straight to `CreateThread`. Your class then would look like this:

```
class Sample {
 DWORD __stdcall Background()
 {
   ... do stuff ...
 }
 void DoSomething()
 {
   ... CreateThread(NULL, 0, ThreadProc<Sample, &Sample::Background>, this, 0,
&dwTid); ...
 }
};
```

This takes the trampoline function out of the class declaration. Instead, it is auto-generated by the compiler via the template.

Okay, so maybe this isn't much of a typing-savings after all, considering the rather clumsy expression for the template invocation. I wonder if it can be made simpler.

[Raymond is currently on vacation; this message was pre-recorded.]

[2004 July 31 – fix obvious typos.]

Raymond Chen

Follow