How can I make a callback function a member of my C++ class?

devblogs.microsoft.com/oldnewthing/20140127-00

January 27, 2014

Raymond Chen

Instead of a Little Program today, I'm going to answer a Little Question. This is a common beginner question, but I figure I'll just spell it out right here for posterity.

First of all, you probably noticed that you can't do this:

```
class CountWindows
{
public:
  int CountThem();
private:
  BOOL CALLBACK WndEnumProc(HWND hwnd, LPARAM lParam);
  int m_count;
};
BOOL CountWindows::WndEnumProc(HWND hwnd, LPARAM lParam)
{
   m_count++;
   return TRUE;
}
int CountWindows::CountThem()
{
  m_count = 0;
  EnumWindows(WndEnumProc, 0); // compiler error here
  return m_count;
}
```

That's because the `WNDENUMPROC` is declared as a so-called *free function*, but member functions are not free. Neither are *function objects* (also known as *functors*) so you can't use a `boost::function` as a window procedure either. The reason is that member functions and functors need to have a hidden `this` parameter, but free functions do not have a hidden `this` parameter.

On the other hand, static methods are free functions. They can get away with it because they don't have a hidden `this` parameter either.

Win32 has a general principle that callback functions have a special parameter where you can pass any information you like (known as *context* or *reference data*), and that same value is passed back to your callback function so it knows what's going on. In practice, most people will pass a pointer to a class or structure.

In other words, the reference data parameter makes explicit what C++ hides (the `this` parameter).

```
class CountWindows
{
public:
  int CountThem();
private:
  static BOOL CALLBACK StaticWndEnumProc(HWND hwnd, LPARAM lParam);
  int m_count;
};
BOOL CountWindows::StaticWndEnumProc(HWND hwnd, LPARAM lParam)
{
   CountWindows *pThis = reinterpret_cast<CountWindows *>(lParam);
   pThis->m_count++;
   return TRUE;
}
int CountWindows::CountThem()
{
  m_count = 0;
  EnumWindows(StaticWndEnumProc, reinterpret_cast<LPARAM>(this));
  return m_count;
}
```

What we did was pass our `this` parameter explicitly as the reference data to the `EnumWindows` function, and then in the callback, cast the reference data back to `this` so that we can use it to access our member variables.

If the `WndEnumProc` is long, then it can get tedious typing `pThis->` in front of everything, so a common follow-up technique is to make the static member function a wrapper that calls a normal member function.

```
class CountWindows
{
public:
  int CountThem();
private:
  static BOOL CALLBACK StaticWndEnumProc(HWND hwnd, LPARAM lParam);
  BOOL WndEnumProc(HWND hwnd);
  int m_count;
};
BOOL CountWindows::StaticWndEnumProc(HWND hwnd, LPARAM lParam)
{
   CountWindows *pThis = reinterpret_cast<CountWindows* >(lParam);
   return pThis->WndEnumProc(hwnd);
}
BOOL CountWindows::WndEnumProc(HWND hwnd)
{
    m_count++;
    return TRUE;
}
int CountWindows::CountThem()
{
  m_count = 0;
  EnumWindows(StaticWndEnumProc, reinterpret_cast<LPARAM>(this));
  return m_count;
}
```

Observe that by putting all the real work inside the traditional member function `CountWindows::WndEnumProc`, we avoid having to type `pThis->` in front of everything.

This principle of using reference data to pass context through a callback is very common in Windows programming. We'll see a few more examples in the future, but I'm not going to jam all the beginner articles in a row because that would bore my regular readers.

Historical note: The term *reference data* was used in 16-bit Windows, but the Windows NT folks preferred to use the term *context*. You can tell which team introduced a particular callback function by seeing what they call that extra parameter.

Raymond Chen

Follow