How can I make a WNDPROC or DLGPROC a member of my C++ class?

devblogs.microsoft.com/oldnewthing/20140203-00

February 3, 2014

Raymond Chen

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

Common special cases for wanting to use a member function as a callback function are the window procedure and its cousin the dialog procedure. The question, then, is where to put the reference data.

Let's start with window procedures. The `CreateWindow` function and its close friend `CreateWindowEx` let you pass your reference data as the final parameter, prototyped as `LPVOID lpParam`. As noted in the documentation, that value is passed back to the window procedure by the `WM_NCCREATE` and `WM_CREATE` messages as part of the `CREATESTRUCT` structure. One of the first messages passed to a window is `WM_NCCREATE`, so that's where we'll grab the reference data and save it for later.

You can follow along in this simple C++ program: The static window procedure handles the `WM_NCCREATE` message by extracting the `lpCreateParams` from the `CREATESTRUCT` and saving it in the `GWLP_USERDATA` window bytes. That value is a special per-window storage location provided for the benefit of the window procedure, and most people use it to store their context parameter for safekeeping.

If the message is not `WM_NCCREATE`, then we retrieve the context parameter from where we had stashed it.

Either way, we end up with a copy of the context parameter. If you want your window procedure to be a member function, the natural choice for the context parameter is the `this` pointer for the instance. The static window procedure therefore tends to look like this:

```
LRESULT CALLBACK MyWindowClass::s_WndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 MyWindowClass *pThis; // our "this" pointer will go here
 if (uMsg == WM_NCCREATE) {
  // Recover the "this" pointer which was passed as a parameter
  // to CreateWindow(Ex).
  LPCREATESTRUCT lpcs = reinterpret_cast<LPCREATESTRUCT>(lParam);
  pThis = static_cast<MyWindowClass*>(lpcs->lpCreateParams);
  // Put the value in a safe place for future use
  SetWindowLongPtr(hwnd, GWLP_USERDATA,
                   reinterpret_cast<LONG_PTR>(pThis));
 } else {
  // Recover the "this" pointer from where our WM_NCCREATE handler
  // stashed it.
  pThis = reinterpret_cast<MyWindowClass*>(
              GetWindowLongPtr(hwnd, GWLP_USERDATA));
 }
 if (pThis) {
  // Now that we have recovered our "this" pointer, let the
  // member function finish the job.
  return pThis->WndProc(hwnd, uMsg, wParam, lParam);
 }
 // We don't know what our "this" pointer is, so just do the default
 // thing. Hopefully, we didn't need to customize the behavior yet.
 return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```

You pass the the `this` pointer to `CreateWindow` as the last parameter, so that the window procedure can pick it up.

```
hwnd = CreateWindow(... other parameters..., this);
```

For dialog boxes, you can do basically the same thing. It's just that the bookkeeping is slightly different.

- The `...Param` versions of the dialog box functions are the ones which let you pass reference data.
- The dialog procedure receives the reference data in the `lParam` passed with the `WM_INITDIALOG`.
- The system-provided secret hiding place for dialog boxes is called `DWLP_USER`.

```
INT_PTR CALLBACK MyDialogClass::s_DlgProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 MyDialogClass *pThis; // our "this" pointer will go here
 if (uMsg == WM_INITDIALOG) {
  // Recover the "this" pointer which was passed as the last parameter
  // to the ...Dialog...Param function.
  pThis = reinterpret_cast<MyDialogClass*>(lParam);
  // Put the value in a safe place for future use
  SetWindowLongPtr(hdlg, DWLP_USER,
                   reinterpret_cast<LONG_PTR>(pThis));
 } else {
  // Recover the "this" pointer from where our WM_INITDIALOG handler
  // stashed it.
  pThis = reinterpret_cast<MyDialogClass*>(
              GetWindowLongPtr(hdlg, DWLP_USER));
 }
 if (pThis) {
  // Now that we have recovered our "this" pointer, let the
  // member function finish the job.
  return pThis->DlgProc(hwnd, uMsg, wParam, lParam);
 }
 // We don't know what our "this" pointer is, so just do the default
 // thing. Hopefully, we didn't need to customize the behavior yet.
 return FALSE; // returning FALSE means "do the default thing"
}
```

The above code should look really familiar, since it's the same as the window procedure case, just with slightly different bookkeeping.

The resulting classes look like this:

```
class MyWindowClass
{
 ... other class stuff goes here ...
 // This is the static callback that we register
 static LRESULT CALLBACK s_WndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
 // The static callback recovers the "this" pointer and then
 // calls this member function.
 LRESULT WndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
};
void MyWindowClass::SomeMemberFunction()
{
 // to register the class
 WNDCLASS wc;
 ... fill out the window class as normal ...
 wc.lpfnWndProc = MyWindowClass::s_WndProc;
 wc.lpszClassName = TEXT("MyWindowClass");
 RegisterClass(&wc);
 // to create a window
 hwnd = CreateWindow(TEXT("MyWindowClass"),
                     ... other parameters as usual ...,
                     this);
}
class MyDialogClass
{
 ... other class stuff goes here ...
 // This is the static callback that we register
 static INT_PTR CALLBACK s_DlgProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam);
 // The static callback recovers the "this" pointer and then
 // calls this member function.
 INT_PTR DlgProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam);
};
void MyDialogClass::SomeMemberFunction()
{
 // to create the dialog box
 DialogBoxParam(...  other parameters as usual ...,
                reinterpret_cast<LPARAM>(this));
}
```

Okay, I'll try to write something more interesting for next week. But at least I wrote this part down so I can point people at it in the future.

Bonus chatter: As commenter Ben noted last week, DDEML is another component that uses the implicit reference data model. In the DDEML case, you use `DdeSetUserHandle` to set the reference data and `DdeQueryConvInfo` to retrieve it.

(Various errors have been corrected based on comments, thanks everybody!)

Raymond Chen

Follow