TMemberCallback: A C++ Class to Implement Member Callback Functions

Version 1.0

Introduction

Have you spent for ever trying to learn C++ really well? Have you spent forever trying to program in Microsoft Windows really well? Have you always wondered then why it is impossible to register a member function of a class as a Windows callback function? The reason is that Windows, as object oriented as it may seem, was not designed for C++. However, with this class that I provide, it is now possible to use C++ member functions as callback functions under Windows 3.1.

Suppose that you did want to call MakeProcInstance() with a C++ member function. Well, first of all, you would have to cast the member function to a FARPROC. Most compilers are wise enough not to let you do such a thing. Even if you could, and even if you could continue as if it would work, the system would come to screeching halt sooner or later. The reason is that each member function call requires a hidden parameter: the hidden `this` pointer. Windows does not know about this hidden pointer. What would be nice would be a way to tell MakeProcInstance() what the `this` pointer is. That is exactly the service that this class provides.

What's Included

You are provided with the necessary items to get you started using the TMemberCallback class. First is the MEMDLL.DLL file. This contains the class which you will inherit from. The second is the MEMCALL.H file. This is the header file which you will be needing for compile time. Third is the MEMDLL.LIB file which you will need to link with. That's all you need, along with this documentation of course.

How it Works

The way this class works is through the beauty of inheritance. All that the user needs to do is to inherit from the TMemberCallback class. With C++, all this requires is the concatenation of the TMemberCallback class to the inheritance list of the class declaration. The class then inherits a class specific version of the MakeProcInstance() function which knows the hidden `this` pointer!

The class specific MakeProcInstance() function does exactly what the global MakeProcInstance() does, and some. The global version of MakeProcInstance() merely loads the `ax` register with the value of instance handle and then jumps to the FARPROC it was provided. The class specific version does this but also makes sure that the `this` pointer is put onto the stack for the member function that it then jumps to. That's it! However, it is easier said than done.

How to Use the TMemberCallback Class

As mentioned previously, the user simply derives his or her class from the TMemberCallback class. For instance if you had a class called TWindow as follows:

```
class TMyWindow : public TWindow
```

```
{

}
```

You would simply have to add the TMemberCallback class to the inheritance list as follows:

```
class TMyWindow : public TWindow, public TMemberCallback
{

}
```

Then, at the appropriate time, the user would call the class specific version of the MakeProcInstance() function. The function takes two parameters, a CALLBACKPROC union and an HINSTANCE. The function prototype is a follows:

```
FARPROC MakeProcInstance(CALLBACKPROC CallbackProc,
                                        HINSTANCE hInstance);
```

The purpose of the CALLBACKPROC union is so that the function can take one parameter which may represent a pointer function to any one of the callback types. Let us suppose that the user wants to register a function `TMyWindow::TimerProc` as a member callback function. The user wishes to create a one second timer. Then the user would take the following action.

```
CALLBACKPROC CallbackProc;
CallbackProc.lpfnTimerProc = (LPFNTIMERPROC)
                                            &TMyWindow::TimerProc;
lpfnTimerProc = MakeProcInstance(CallbackProc, hInstance);
SetTimer(NULL, NULL, 1000, lpfnTimerProc);
```

Note: The pointer to the member function that the programmer wishes to use as a callback **must** be typecast into the corresponding pointer type. This is necessary in order to convert the pointer into a TMemberCallback member function pointer.

The previous example conveys the main idea. The programmer may also want to monitor the return value from MakeProcInstance() and make sure that it is not NULL. A NULL return value indicates a memory allocation failure. To be complete, I have added a class specific version of FreeProcInstance() also. The call to FreeProcInstance() is not necessary since the class keeps track of the instance procedures and deletes them when the class is destructed. However, it may be necessary for when a class derived from TMemberCallback needs to manipulate callback functions often.

As one can see, it is very easy to use this class! Simply add the include file, inherit from the class, and link with the required .LIB file, and voila!

Technical Aspects of the TMemberCallback Class

The idea is simple. For Windows to call a function as a callback function, it must already know what the function "looks like." In other words, the parameters passed and values returned must follow a strict template. This is the hard reality about using callback functions. It is possible to trick the compiler at compile time and tell Windows to call a function that does not match the template for the particular callback function. However, it would most certainly cause an instant General Protection Fault which we all know and love.

This very aspect is what causes C++ member functions to be invalid as Windows callback functions. Because even if you declare them the way they must be declared, there is still one oversight: The hidden `this` pointer. Windows does not know about it. Therefore, even if you could trick the compiler into letting you use one as a callback, the

program would fail. Many people get around this road block by using static member functions. I personally feel that this is a kludge. It also causes two problems. The first is that the static member function is not passed a `this` pointer. The function does not know the object which it works on, unless, of course, you pass it the pointer some way. Another problem is that there can only be one callback routine for the whole set of objects of that particular class due to the function being static.

A better way to do it is this. Why not trick Windows into thinking that it knows about the hidden `this` pointer? This is relatively easy. When you call MakeProcInstance(), you are usually using it before registering a callback function so that the `ds` register will be properly set when the callback function is called. What it does is create a code stub in memory which loads the `ax` register with the proper value and then jumps to your callback. You then tell Windows to call the proc instance that you just created instead of the actual function. Why not just take care of the hidden `this` pointer at the same time? All the code stub would have to do is adjust the stack so that the pointer would be on the stack when the stub jumped to the original proc instance (which in our case would be a member function).

The TMemberCallback class provides you with a class specific version of MakeProcInstance() and FreeProcInstance(). When you call this version of MakeProcInstance(), it creates a code stub which adjusts the stack, loads the `ax` register, and then jumps to the callback routine. You just register this code stub as the proc instance with Windows! It's that simple.

Note: This function assumes that the hidden `this` pointer is a LONG pointer. This is usually nothing to worry about. However, if there is a compiler switch which forces the compiler to use LONG `this` pointers, I would suggest setting it accordingly.

The parameters that MakeProcInstance() takes are a CALLBACKPROC data type and an HINSTANCE data type. The programmer must first declare a CALLBACKPROC data type before registering the callback function. This is a union which contains a function pointer. It is necessary to facilitate the casting of a member function pointer to a regular function pointer. Normally compilers will not allow this, but it must be done in order to create the code stub. The CALLBACKPROC data type allows you to load it with any type of member callback function it knows about. The header file should be examined in order to see if the appropriate callback function is supported. If I have left one out, please contact me at the address at the end of this manual. MakeProcInstance() returns a FARPROC which is what needs to be registered with Windows.

Note: Remember to typecast the member function pointer using one of the given typecasts in the header file. This is required in order to convert the pointer to a TMemberCallback member function pointer.

For completeness sake, I have included a class specific FreeProcInstance() function. In some cases, it might be necessary to use it. However, for a program which registers only one or two callbacks during its lifetime, it may not need to be used. The TMemberCallback class keeps track of all of the code stubs that have been created. When the object is destroyed, it automatically gets rid of these code stubs and sends them the way of the dodo. However, **don't forget to unregister your callback functions before program termination!!!!**

Compiler Dependencies

This code was originally compiled under Borland C++ 3.1 for use with OWL. However, I have tried to make it compatible with different compilers in mind. There might be a couple of keywords that other compilers might choke on. One of them is the CLASSDEF macro. It is an OWL thing which automatically creates typedefs for you to reference your objects. For instance, if I create a class called OBJECT, it will automatically create typedefs for me to address it using pointers and references by creating a POBJECT and ROBJECT type respectively. This is easy to work around. Simply declare the typedefs yourself (don't forget to make them far pointers and references).

The only other thing that I can think of off of the top of my head is the _EXPORT keyword in the class declaration of the TMemberCallback class. In OWL, it expands to the type of memory model which is currently in use (far, huge, etc.). However, you must always try to use far or huge since the `this` pointer must always be a LONG pointer as noted previously.

Any Questions... Comments?

If you have any questions or comments, please feel free to contact me. I can be reached via email at the following:

tnash@azariah.tamu.edu

If you like this utility and plan to use it a lot, please send $10 or what you consider a reasonable donation for the many hours spent in front of a debugging monitor figuring out the intricacies of this process. Please send donations to:

Trey Nash
5016 Forest Bend
Dallas, Texas 75244

Thank you for your support.