The new scratch program

devblogs.microsoft.com/oldnewthing/20050422-08

April 22, 2005

Raymond Chen

I think it's time to update the scratch program we've been using for the past year. I hear there's this new language called C++ that's going to become really popular any day now, so let's hop on the bandwagon!

```
#define STRICT
#define UNICODE
#define _UNICODE
#include <windows.h>
#include <windowsx.h>
#include <ole2.h>
#include <commctrl.h>
#include <shlwapi.h>
#include <shlobj.h>
#include <shellapi.h>
HINSTANCE g_hinst;
class Window
{
public:
 HWND GetHWND() { return m_hwnd; }
protected:
 virtual LRESULT HandleMessage(
                        UINT uMsg, WPARAM wParam, LPARAM lParam);
 virtual void PaintContent(PAINTSTRUCT *pps) { }
 virtual LPCTSTR ClassName() = 0;
 virtual BOOL WinRegisterClass(WNDCLASS *pwc)
     { return RegisterClass(pwc); }
 virtual ~Window() { }
 HWND WinCreateWindow(DWORD dwExStyle, LPCTSTR pszName,
       DWORD dwStyle, int x, int y, int cx, int cy,
       HWND hwndParent, HMENU hmenu)
 {
  Register();
  return CreateWindowEx(dwExStyle, ClassName(), pszName, dwStyle,
                  x, y, cx, cy, hwndParent, hmenu, g_hinst, this);
 }
private:
 void Register();
 void OnPaint();
 void OnPrintClient(HDC hdc);
 static LRESULT CALLBACK s_WndProc(HWND hwnd,
     UINT uMsg, WPARAM wParam, LPARAM lParam);
protected:
 HWND m_hwnd;
};
void Window::Register()
{
    WNDCLASS wc;
    wc.style         = 0;
    wc.lpfnWndProc   = Window::s_WndProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = g_hinst;
    wc.hIcon         = NULL;
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wc.lpszMenuName  = NULL;
```

```
    wc.lpszClassName = ClassName();
    WinRegisterClass(&wc);
}
LRESULT CALLBACK Window::s_WndProc(
               HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 Window *self;
 if (uMsg == WM_NCCREATE) {
  LPCREATESTRUCT lpcs = reinterpret_cast<LPCREATESTRUCT>(lParam);
  self = reinterpret_cast<Window *>(lpcs->lpCreateParams);
  self->m_hwnd = hwnd;
  SetWindowLongPtr(hwnd, GWLP_USERDATA,
            reinterpret_cast<LPARAM>(self));
 } else {
  self = reinterpret_cast<Window *>
            (GetWindowLongPtr(hwnd, GWLP_USERDATA));
 }
 if (self) {
  return self->HandleMessage(uMsg, wParam, lParam);
 } else {
  return DefWindowProc(hwnd, uMsg, wParam, lParam);
 }
}
LRESULT Window::HandleMessage(
                          UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 LRESULT lres;
 switch (uMsg) {
 case WM_NCDESTROY:
  lres = DefWindowProc(m_hwnd, uMsg, wParam, lParam);
  SetWindowLongPtr(m_hwnd, GWLP_USERDATA, 0);
  delete this;
  return lres;
 case WM_PAINT:
  OnPaint();
  return 0;
 case WM_PRINTCLIENT:
  OnPrintClient(reinterpret_cast<HDC>(wParam));
  return 0;
 }
 return DefWindowProc(m_hwnd, uMsg, wParam, lParam);
}
void Window::OnPaint()
{
 PAINTSTRUCT ps;
 BeginPaint(m_hwnd, &ps);
 PaintContent(&ps);
 EndPaint(m_hwnd, &ps);
}
void Window::OnPrintClient(HDC hdc)
{
 PAINTSTRUCT ps;
```

```
 ps.hdc = hdc;
 GetClientRect(m_hwnd, &ps.rcPaint);
 PaintContent(&ps);
}
class RootWindow : public Window
{
public:
 virtual LPCTSTR ClassName() { return TEXT("Scratch"); }
 static RootWindow *Create();
protected:
 LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam);
 LRESULT OnCreate();
private:
 HWND m_hwndChild;
};
LRESULT RootWindow::OnCreate()
{
 return 0;
}
LRESULT RootWindow::HandleMessage(
                         UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
  case WM_CREATE:
   return OnCreate();
  case WM_NCDESTROY:
   // Death of the root window ends the thread
   PostQuitMessage(0);
   break;
  case WM_SIZE:
   if (m_hwndChild) {
    SetWindowPos(m_hwndChild, NULL, 0, 0,
                 GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam),
                 SWP_NOZORDER | SWP_NOACTIVATE);
   }
   return 0;
  case WM_SETFOCUS:
   if (m_hwndChild) {
    SetFocus(m_hwndChild);
   }
   return 0;
 }
 return __super::HandleMessage(uMsg, wParam, lParam);
}
RootWindow *RootWindow::Create()
{
 RootWindow *self = new RootWindow();
 if (self && self->WinCreateWindow(0,
       TEXT("Scratch"), WS_OVERLAPPEDWINDOW,
       CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
       NULL, NULL)) {
      return self;
```

```
  }
 delete self;
 return NULL;
}
int PASCAL
WinMain(HINSTANCE hinst, HINSTANCE, LPSTR, int nShowCmd)
{
 g_hinst = hinst;
 if (SUCCEEDED(CoInitialize(NULL))) {
  InitCommonControls();
  RootWindow *prw = RootWindow::Create();
  if (prw) {
   ShowWindow(prw->GetHWND(), nShowCmd);
   MSG msg;
   while (GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
   }
  }
  CoUninitialize();
 }
 return 0;
}
```

The basic idea of this program is the same as our old scratch program, but now it has that fresh lemony C++ scent. Instead of keeping our state in globals, we declare a C++ class and hook it up to the window. For simplicity, the object's lifetime is tied to the window itself.

First, there is a bare-bones `Window` class which we will use as our base class for any future "class associated with a window" work. The only derived class for now is the `RootWindow`, the top-level frame window that for now is the only window that the program uses. As you may suspect, we may have other derived classes later as the need arises.

The reason why the `WinRegisterClass` method is virtual (and doesn't do anything interesting) is so that a derived class can modify the `WNDCLASS` that is used when the class is registered. I don't have any immediate need for it, but it'll be there if I need it.

We use the `GWLP_USERDATA` window long to store the pointer to the associated class, thereby allowing us to recover the object from the window handle.

Observe that in the `RootWindow::HandleMessage` method, I used the Visual C++ `__super` extension. If you don't want to rely on a nonstandard extension, you can instead write

```
class RootWindow : public Window
{
public:
 typedef Window super;
 ...
```

and use `super` instead of `__super`.

This program doesn't do anything interesting; it's just going to be a framework for future samples.

Raymond Chen

Follow