System Modal Back

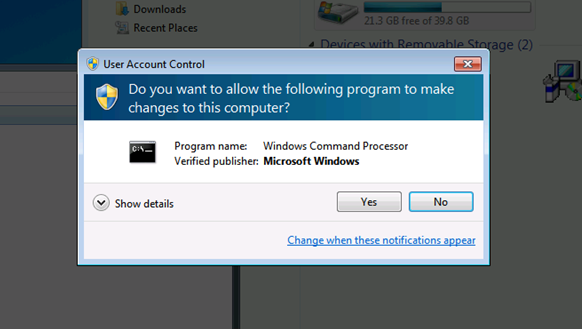
*This article makes introduction to the one of the interesting Windows abilities, the ability to deal with multiple desktops. As example it contains the source of code of the .NET library that gives ability of creation System Modal dialogs in the same way as UAC window does. The article is for beginners and requires only very basic knowledge about.NET and Windows API.*

# Introduction

Very old operation systems like Windows 3.11, Windows 95/98/ME supported a special window style WS\_SYSMODAL. Window with such style blocks the entire GUI until user will close the window. But Microsoft decides that system modal windows contradict the concept of multitasking and since Windows NT 4.0 Microsoft doesn’t allow to create system modal windows.

However there is certain kind of situations when application needs to prevent user’s activity or restrict it for example allow working only with one special window. It could be security applications, applications that play back some macro or even applications that intentionally block user’s activity for his safety.

All of you have seen UAC dialogs in Windows Vista/2008/Seven. For me that window looks exactly like “System Modal” window should. Once it pops up other windows are down lighted and you can’t switch to them until UAC window will be closed.



Want to create same dialog in your application? No problem. Here is a [link](http://www.google.com) to the .NET assembly with only one public method ShowSystemModalDialog.

To create such a system modal dialog you should call this method:

**Developex.ScreenLocker.ShowSystemModalDialog("MyTestDesktop", new MyForm());**

Instead:

**new MyForm().ShowDialog();**

|  |
| --- |
| Warning |
| Pay attention that ScreenLocker assembly is a mixed mode assembly that contains unmanaged 32 bits code. With default Platform Target setting (Any CPU) of any .Net project application will not be able to run on 64 bits operations systems. To fix this change Platform Target to “x86”. |

Want to know how you can create such a dialog from your application?

OK. But actually this question isn’t very correct because UAC dialog is absolutely regular Windows dialog. The only difference is that it runs into a dedicated desktop. And all other windows are looking dark because of illusion. The background of the desktop with UAC windows is made from the screenshot of your default desktop.

Therefore to create system modal window we need to know how to:

1. Switch the system into another desktop.
2. Create windows into the new desktop.
3. Make screenshot and set it as a background of another desktop.

# Switch into another desktop

This is the simplest part. There is a function in Windows API SwitchDesktop doing that. This is a rather simple function it inputs only one parameter – a handle to the desktop. But to switch to the desktop we need to create it first. Function CreateDesktop creates a desktop.

For example such code on C++ creates and switches the system to the new desktop for 5 seconds:

**HDESK hDeskOld = GetThreadDesktop(GetCurrentThreadId());**

**HDESK hDesk = CreateDesktop(L"MyDesktop", NULL, NULL, 0, GENERIC\_ALL, NULL);**

**SwitchDesktop(hDesk);**

**Sleep(5000);**

**SwitchDesktop(hDeskOld);**

**CloseDesktop(hDesk);**

|  |
| --- |
| Advice |
| During writing a code that switches system to another desktop you can easily get into the situation when you can’t switch back to your default desktop. This is not very nice because then you will need to reboot your system. Actually logoff and re-logon is enough, but it also not very nice to restart the development environment every time you do a mistake. To prevent such situation I suggest you to write a very small application like this:  // SwitchDesktop.cpp : Defines the entry point for the console application.  #include <Windows.h>  int \_tmain(int argc, \_TCHAR\* argv[])  {  SwitchDesktop(OpenDesktop(argv[1], 0, FALSE, DESKTOP\_SWITCHDESKTOP));  return 0;  }  Then you always could run command “SwitchDesktop.exe Default” from the Task Manager. Yes, you can run Task Manager into the new desktop by pressing Ctrl + Shift + Esc too. |

On Windows Vista or newer OS CreateDesktop works fine but on Windows XP unfortunately it does not always. It’s because to create a desktop operation system must allocate some amount of physical defragmented area of memory. On XP if there is no appropriate defragmented area of the physical memory at the moment then function fails. Therefore with some probability this function could not work as we expect. This probability depends from many factors but it’s big enough and we can’t just ignore it.

Fortunately there are some solutions how we can minimize this probability. The simplest way is create a desktop during system startup in some Windows service. At that time OS must have defragmented areas of physical memory for sure. So if for some reason you still need to support XP for your application – be aware it’s possible.

# Create a window on the new desktop

Every thread is always related to the one desktop. When we create a window it is always created on the desktop associated with the thread creating the window. Function SetThreadDesktop can reassign the thread desktop. But this function fails if the calling thread has some windows or hooks on its current desktop. Therefore you can’t create 2 windows on different desktops from one thread.

Here is a small class written on managed C++ that could be used to run some dialog in the dedicated desktop.

**// ScreenLocker.h**

**#pragma once**

**using namespace System;**

**using namespace System::Windows::Forms;**

**namespace Developex**

**{**

**public ref class ScreenLocker**

**{**

**private:**

**String ^\_desktopName;**

**Form ^\_form;**

**void DialogThread(void);**

**public:**

**static void ShowSystemModalDialog (String ^desktopName, Form ^form);**

**};**

**}**

**// ScreenLocker.cpp**

**#include "stdafx.h"**

**#include "ScreenLocker.h"**

**using namespace System::Threading;**

**using namespace System::Runtime::InteropServices;**

**namespace Developex**

**{**

**void ScreenLocker::DialogThread()**

**{**

**// Save the handle to the current desktop**

**HDESK hDeskOld = GetThreadDesktop(GetCurrentThreadId());**

**// Create a new desktop**

**IntPtr ptr = Marshal::StringToHGlobalUni(\_desktopName);**

**HDESK hDesk = CreateDesktop((LPCWSTR)ptr.ToPointer(),**

**NULL, NULL, 0, GENERIC\_ALL, NULL);**

**Marshal::FreeHGlobal(ptr);**

**// Switch to the new deskop**

**SwitchDesktop(hDesk);**

**// Assign new desktop to the current thread**

**SetThreadDesktop(hDesk);**

**// Run the dialog**

**Application::Run(\_form);**

**// Switch back to the initial desktop**

**SwitchDesktop(hDeskOld);**

**CloseDesktop(hDesk);**

**}**

**void ScreenLocker::ShowSystemModalDialog(String ^desktopName, Form ^form)**

**{**

**// Create and init ScreenLocker instance**

**ScreenLocker ^locker = gcnew ScreenLocker();**

**locker->\_desktopName = desktopName;**

**locker->\_form = form;**

**// Create a new thread for the dialog**

**(gcnew Thread(gcnew ThreadStart(locker,**

**&Developex::ScreenLocker::DialogThread)))->Start();**

**}**

**}**

As you can see ScreenLocker class contains only one public method ShowSystemModalDialog. As written above to run you dialog on the dedicated desktop write:

**ScreenLocker.ShowSystemModalDialog("MyDesktop", new MyForm());**

# Create a background on a desktop

The latest step is to put some graphics effects on the background of our desktop. To put something on the background we can create a disabled window to full screen. A disabled window ignores mouse clicks and cannot be selected via Alt-Tab. As you probably know to create a window using Windows API we need to create a Window Class, define window procedure and call CreateWindows function. Here is the example of code that creates such a window:

**// Window procedure of the background window**

**LRESULT CALLBACK BackgroundWindowProc(HWND hWnd, UINT nMessage, WPARAM wParam, LPARAM lParam)**

**{**

**LRESULT res = FALSE;**

**switch (nMessage)**

**{**

**// Implament WM\_PAINT message handler**

**case WM\_PAINT:**

**{**

**PAINTSTRUCT ps = {0};**

**HDC hDC = BeginPaint(hWnd, &ps);**

**RECT rcDesktop = {0};**

**GetWindowRect(GetDesktopWindow(), &rcDesktop);**

**BitBlt(hDC, 0, 0, rcDesktop.right, rcDesktop.bottom,**

**hBKDC, 0, 0, SRCCOPY);**

**EndPaint(hWnd, &ps);**

**res = TRUE;**

**}**

**break;**

**// Ignore WM\_CLOSE event**

**case WM\_CLOSE:**

**break;**

**default:**

**res = DefWindowProc(hWnd, nMessage, wParam, lParam);**

**}**

**return res;**

**}**

**// Create Window Class**

**WNDCLASS wc = {0};**

**wc.lpfnWndProc = (WNDPROC)ScreenLockerPrivate::BackgroundWindowProc;**

**wc.hInstance = (HINSTANCE)Process::GetCurrentProcess()->Handle.ToPointer();**

**wc.lpszClassName = L"DevelopexScreenLockerBK";**

**RegisterClass(&wc);**

**// Create Background Window**

**RECT rcDesktop = {0};**

**GetWindowRect(GetDesktopWindow(), &rcDesktop);**

**hBKWindow = CreateWindow(L"DevelopexScreenLockerBK", L"",**

**WS\_VISIBLE | WS\_POPUP | WS\_DISABLED,**

**rcDesktop.left, rcDesktop.top, rcDesktop.right, rcDesktop.bottom,**

**NULL, NULL, wc.hInstance, NULL);**

The only thing the current window does is draw itself. The red line takes calls BitBlt function to draw some saved bitmap over the window. The code bellow grabs the current desktop to the temporary bitmap:

**// Retrive desktop size**

**RECT rcDesktop = {0};**

**GetWindowRect(GetDesktopWindow(), &rcDesktop);**

**HDC hDesktopDC = GetDC(GetDesktopWindow());**

**// Prepeare context for background**

**hBKDC = CreateCompatibleDC(hDesktopDC);**

**hBitmapBKOld = SelectObject(hBKDC, CreateCompatibleBitmap(hDesktopDC, rcDesktop.right,**

**rcDesktop.bottom));**

**// Grab a screenshot from current desktop**

**BitBlt(hBKDC, 0, 0, rcDesktop.right, rcDesktop.bottom, hDesktopDC, 0, 0, SRCCOPY);**

You can reduce lighting of the image by drawing another semitransparent black bitmap with AlphaBlend function.

**// Reduce lighting**

**HDC hBKDC2L = CreateCompatibleDC(hDesktopDC);**

**HGDIOBJ hBitmapBK2LOld = SelectObject(hBKDC2L, CreateCompatibleBitmap(hDesktopDC, rcDesktop.right, rcDesktop.bottom));**

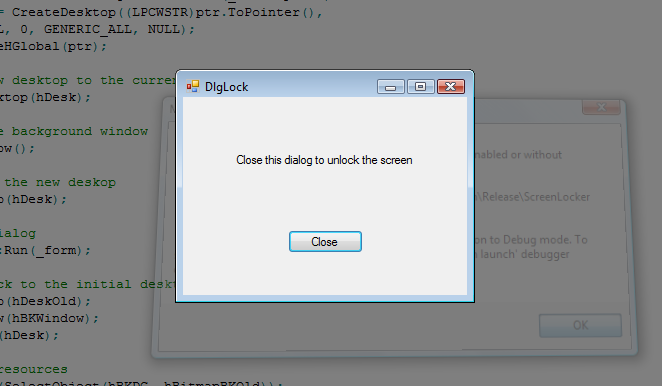
**ReleaseDC(GetDesktopWindow(), hDesktopDC);**

**FillRect(hBKDC2L, &rcDesktop, (HBRUSH)GetStockObject(BLACK\_BRUSH));**

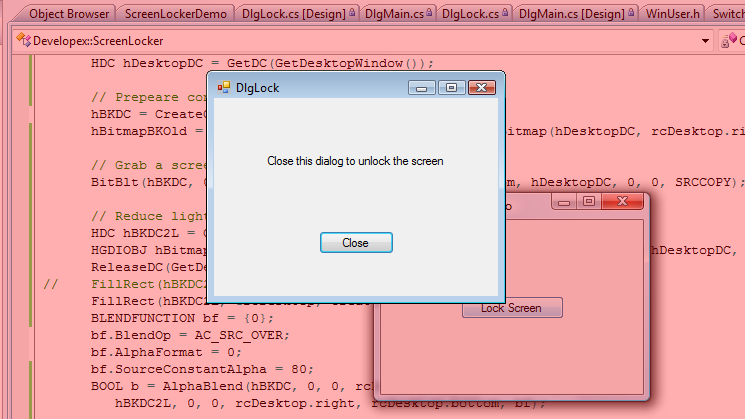
**BLENDFUNCTION bf = {AC\_SRC\_OVER, 0, 128};**

**BOOL b = AlphaBlend(hBKDC, 0, 0, rcDesktop.right, rcDesktop.bottom,**

**hBKDC2L, 0, 0, rcDesktop.right, rcDesktop.bottom, bf);**



Or not a black bitmap:



I’m sure you can feign a lot of interesting ways how to change the background.

And don’t forget to free all GDI resources.

**// Free GDI resources**

**DeleteObject(SelectObject(hBKDC2L, hBitmapBK2LOld));**

**DeleteObject(hBKDC2L);**

# Conclusion

As we found out despite that Windows doesn’t support WS\_SYSMODAL has all necessary functionally to create system modal dialogs. But general I agree with Microsoft. System modal dialogs contradict the concept of multitasking and UAC window is probably just an exception:)

*Mike Makarov*

*Developex CTO*