

## Subclassing Windows Explorer for Context Menu Overriding / PInvoke / C#

### Subclassing Windows Explorer for Context Menu Overriding / PInvoke / C#

This article has been put together to fill a void that I found when trying to use the Windows Explorer object. I have recently undertaken a project at work that required me to embed a browser object into an application and display reports through Microsoft Reporting Services.

Add an item to Windows Explorer context (right-click) menu easily &ndash; How ?

Add items to Explorer Shell context menu easily with Windows Explorer Shell Context Menu. This powerful .Net component for custom items adding to Windows Explorer Shell context menu will add all your entries to Explorer context menu. This .Net component with full C# and VB.NET support include detailed C# / VB.NET samples, tutorials , user-friendly manuals and support all you may need to add your items to context menu :

- Add items to Windows Explorer Shell context menu to be shown on any Windows computer (all operating systems are supported &ndash; Windows XP, Vista, Windows x64 , etc.)
- Add all your items to Windows Explorer Shell context menu to be shown in any way - with custom caption and your custom icon, as separator or sub-menu
- Add your items to Windows Explorer Shell context menu to be shown for all files or shown only for computer files of particular type (for example, only for .PDF .TXT , .MP3,.WMA,.AAC , .WMV media files)
- Add your program entries to Windows Explorer Shell context menu, sub-menus, sub-menus of unlimited depth and even much more

Explorer Context Menu - is a powerful .Net component that support all you need to insert your program items to Windows Explorer Shell context menu - in a fast and a very easy way. Add all your application entries to Windows Explorer Shell context menu right now &ndash; add items to context menu fast , easy and exactly as you want :

This article describes how to add entries to Windows Explorer right-click context menu, custom entries appending method describes here works only for Windows 95 / Windows 98 (not on XP, Vista, x64 - 64-bit Windows), so to add items to Windows Explorer Shell context menu you should use, according to Microsoft guidelines, appropriate .Net component - Windows Explorer Shell Context Menu. This .Net component will add all your custom items to Windows Explorer Shell context menu.

There was a need to stop the user from making use of the Windows through the software, but also to stop them from looking at the underlying code as well as giving them extra options on the context menu.

### Background

The Windows Explorer is an odd control. Essentially if you use a special software you can see that the object has children that protect the object from people picking off the handle to the real Windows Explorer easily. There was a need to iterate through the browser object looking for the child object called Windows Explorer\_Server .

Much of the child search is done in the API Call 'EnumChildWindow'. The snippet of code makes use of some API calls to capture the correct handle for the Windows Explorer and then override the Context Menu and replace it with one that I wanted the user to see.

### Win32 API in C# & Delegates

```
private delegate Int32 EnumChildProc(IntPtr hWnd, IntPtr lParam);
private delegate int Win32WndProc(IntPtr hWnd, int Msg,
    IntPtr wParam, IntPtr lParam);
```

```
[DllImport("user32")]
public static extern int GetClassName(int hwnd,
    StringBuilder lpClassName, int nMaxCount);
[DllImport("user32")]
public static extern int EnumChildWindows(int hwndParent,
    Delegate lpEnumFunc, int lParam);
[DllImport("user32")]
private static extern IntPtr SetWindowLong(IntPtr hWnd,
    int nIndex, Win32WndProc newProc);
[DllImport("user32")]
private static extern int CallWindowProc(IntPtr lpPrevWndFunc,
    IntPtr hWnd, int Msg, int wParam, int lParam);
[DllImport("user32")]
private static extern IntPtr SetWindowLong(IntPtr hWnd,
    int nIndex, IntPtr newProc);
[DllImport("user32")]
public static extern int GetCursorPos(ref Point lpPoint);
[DllImport("user32")]
public static extern int ScreenToClient(int hwnd, ref Point lpPoint);
```

To access these external API functions we need the namespace `System.Runtime.InteropServices`. This namespace provides a collection of classes useful for accessing COM objects, and native APIs. Shown below is the way the code is written in C#:

```
[DllImport("user32")]
public static extern int EnumChildWindows(int hwndParent,
    Delegate lpEnumFunc, int lParam);
```

The `extern` keyword indicates that the method is implemented externally. `extern` methods must be declared as `static`.

Within this API call you may notice the `Delegate` reference.

```
private delegate Int32 EnumChildProc(IntPtr hWnd, IntPtr lParam);
```

The delegate needs to be created in the same way that we create a class, shown below.

```
EnumChildProc myEnumChildProc = new EnumChildProc(EnumChild);
EnumChildWindows(hwnd.ToInt32(), myEnumChildProc, hwnd.ToInt32());
```

The first line shows the creation of the delegate, we pass to `EnumChildProc` a reference to the `EnumChild` function. We then use the `EnumChildWindow` API call to drill down into the child windows and compare the name of the window against the one that we are trying to find.

This `CallBack` feature will iterate through each child until it either runs out of children or finds the window. Using the code

To make use of this code and get you on the road to creating your very own web aware app, you need to add the 'IEHwd.cs' class to your project.

Once this is done, you need to create an instance of the object at a global level to your form.

```
private IEHwd o = new IEHwd();
```

Once this is done you can now access the features within. You will need to pass to the class an instance of your form and also the context menu that you want to show when the right click is done.

I use the `NavigateComplete2` event that gets fired once a response has been received from the web site that the page is being produced.

```
if(o.oldWndProc.ToInt32() == 0)
{
    IntPtr s = o.IEFromhWnd(wbBrowser.Handle);

    o.IEContextMenu = IECustomContext;
```

```
o.CurrForm = this;

o.StartSubclass(s);
}
```

The above statement goes into the IEHwd.cs class and retrieves the browser handle. We pass the web browser handle into the object as a starting point. We then pass the Context Menu into the class via a property and the same for the current form.

Once that is done we call StartSubclass where we store away the old handle so that we can exit the program cleanly at the end (when the form is discarded.)

```
switch(Msg)
{
  case WM_RBUTTONDOWN:
  {
    //Right Mouse Button has been pressed
    //We need to capture the area on the screen
    //Next to the mouse we post our Custom Context Menu
    Point p = new Point();
    GetCursorPos(ref p);
    ScreenToClient(currForm.Handle.ToInt32(), ref p);
    ieContextMenu.Show(currForm, p);

    return 0;
  }
  case WM_RBUTTONDBLCLK:
    return 0;
  default:
    break;
}
```

When the user clicks the right mouse button over the Windows Explorer window we want to capture the right click event. This happens by checking for the WM\_RBUTTONDOWN. Once pressed we need to get the point where the mouse is using the the GetCursorPos. Then we make use of the ScreenToClient API call to make sure that the Context Menu appears under the mouse and not somewhere else on the screen.

Finally we show the Context Menu that we passed into the class earlier and return 0 to stop the real Windows Explorer Shell Context Menu from showing.

You may notice that I also trap the WM\_RBUTTONDBLCLK message also. I found if you repeatedly clicked the right mouse button it would display the real Windows Explorer Shell Context Menu.