How can I detect whether the Game Bar is covering my window?

devblogs.microsoft.com/oldnewthing/20160629-00

June 29, 2016

Raymond Chen

Pressing the Win + G hotkey opens the Game Bar, which lets you record game clips and screenshots. Actually, I use it a lot even for programs that aren't games: It's great for taking video clips of a bug.

Anyway, maybe you have a program that wants to know when the Game Bar is on screen. For example, if you're a game, you may want to pause the game automatically when the user is trying to configure their screen capture.

If you are writing a Store app, you can register for Game Bar events. Here's the short version for C# apps:

```
if (Windows.Gaming.UI.GameBar.Visible) {
  the game bar is visible;
}

if (Windows.Gaming.UI.GameBar.IsInputRedirected) {
  the game bar has input;
}

Windows.Gaming.UI.GameBar.VisibilityChanged +=
    (s, e) => { the visibility changed };
Windows.Gaming.UI.GameBar.IsInputRedirectedChanged +=
    (s, e) => { the input state changed };
```

(Of course, you can avoid having to type `Windows.Gaming.UI` all the time by using the `using` statement, but I'm writing it out just to make it explicit what's going on.)

If you are a desktop app, you will have to talk to the ABI. It's not too difficult, although it is a bit more tedious.

Continuing our crash course in projection:

	Call static method

ABI	`IWidgetStatics* widgetStatics;` `GetActivationFactory(L"Widget", &widgetStatics);` `widgetStatics->Foo();`
C++/CX	`Widget::Foo();`
C#	`Widget.Foo();`
JavaScript	`Widget.foo();`

At the ABI level, static members of a Windows Runtime class are represented as instance members of the class's activation factory. By convention, the interface name for static members is the runtime class name, followed by the word `Statics`.

Okay, we now know just enough to be dangerous. Start with the scratch program and make these changes. (Remember, Little Programs do little to no error checking.)

```
#include <wrl/client.h>
#include <wrl/event.h>
#include <wrl/wrappers/corewrappers.h>
#include <windows.gaming.ui.h>
#include <EventToken.h>
#include <tchar.h> // Huh? Why are you still using ANSI?

namespace WRL = Microsoft::WRL;
namespace awf = ABI::Windows::Foundation;
namespace gameui = ABI::Windows::Gaming::UI;

WRL::ComPtr<gameui::IGameBarStatics> g_gameBarStatics;
boolean g_isVisible;
boolean g_isInputRedirected;
EventRegistrationToken g_tokenVisibility;
EventRegistrationToken g_tokenInput;
```

After including a few header files and declaring some namespace aliases, we create a few global variables to keep track of our state. In a real program, these would probably be instance members of some C++ class, but I'm being lazy.

```
void CheckGameBarVisibility(HWND hwnd)
{
    boolean isVisible;
    g_gameBarStatics->get_Visible(&isVisible);
    if (g_isVisible != isVisible)
    {
        g_isVisible = isVisible;
        InvalidateRect(hwnd, nullptr, TRUE);
    }
}

void CheckGameBarInput(HWND hwnd)
{
    boolean isInputRedirected;
    g_gameBarStatics->get_IsInputRedirected(&isInputRedirected);
    if (g_isInputRedirected != isInputRedirected)
    {
        g_isVisible = isVisible;
        InvalidateRect(hwnd, nullptr, TRUE);
    }
}
```

These two little functions read the current visibility and input redirection states of the game bar, and if they changed, we invalidate the window. We learned about property access a little while ago. In our case, the properties are static, so the property accessors live on the `Statics` interface.

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
  Windows::Foundation::GetActivationFactory(WRL::Wrappers::HStringReference(
    RuntimeClass_Windows_Gaming_UI_GameBar).Get(), &g_gameBarStatics);

  auto visibilityHandler = WRL::Callback<awf::IEventHandler<IInspectable*>>(
    [hwnd](IInspectable*, IInspectable*)
    {
      CheckGameBarVisibilty(hwnd);
      return S_OK;
    });
  g_gamebarStatics->add_VisibilityChanged(visibilityHandler.Get(),
&g_tokenVisibility);

  auto inputHandler = WRL::Callback<awf::IEventHandler<IInspectable*>>(
    [hwnd](IInspectable*, IInspectable*)
    {
      CheckGameBarInput(hwnd);
      return S_OK;
    });
  g_gamebarStatics->add_IsInputRedirectedChanged(inputHandler.Get(), &g_tokenInput);

  CheckGameBarVisibility(hwnd);
  CheckGameBarInput(hwnd);
  return TRUE;
}
```

We create the game bar statics by asking for the `IGameBarStatics` interface from the activation factory. From there, we register two event handlers, one to be called when the visibility changes, and another to be called when input redirection changes. In both cases, we respond to the event by checking the new visiblity or input redirection state.

After registering the handlers, we manually check the visibility and input to get the initial values set up properly.

```
void
OnDestroy(HWND hwnd)
{
  g_gameBar->remove_VisibilityChanged(g_tokenVisibility);
  g_gameBar->remove_IsInputRedirectedChanged(g_tokenInput);
  g_gameBar.Reset();
  PostQuitMessage(0);
}
```

Naturally, we need to clean up when we’re done.

```
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
  PCTSTR visibleMessage =
    g_isVisible ? TEXT("GameBar is visible")
                : TEXT("GameBar is not visible");

  TextOut(pps->hdc, 0, 0, visibleMessage, _tcslen(visibleMessage));

  PCTSTR inputMessage =
    g_isInputRedirected ? TEXT("GameBar has taken input")
                        : TEXT("GameBar does not have input");

  TextOut(pps->hdc, 0, 20, inputMessage, _tcslen(inputMessage));
}
```

Our `PaintContent` function prints the current state of the game bar: Is it visible? Does it have input?

And that's it. Run this program, press the `Win`+`G` hotkey to call up the game bar, and observe that the program updates its window to reflect the game bar state.

Raymond Chen

Follow