How can I get notified when some other window is destroyed?

devblogs.microsoft.com/oldnewthing/20111026-00

October 26, 2011

Raymond Chen

A customer wanted to know whether there was a method (other than polling) to monitor another window and find out when it gets destroyed. The goal was to automate some operation, and one of the steps was to wait until some program closed its XYZ window before moving on to the next step. Finding the XYZ window could be done with a `FindWindow`, but since the window belongs to another process, you can't subclass it to find out when it gets destroyed.

Enter accessibility.

The `SetWinEventHook` function lets you monitor accessibility events, and you can do it globally, for a particular process, or for a particular thread. Since we're interested in just one specific window, we can restrict our monitoring to a specific process and thread. (You don't want to monitor too much or you end up getting spammed with notifications you don't care about, which will annoy both you and the end users who are wondering why all their CPU is being consumed on pointless activity.)

Let's take our scratch program and have it monitor an arbitrary window whose name is passed on the command line.

```
HWND g_hwnd; /* our main window */
HWND g_hwndTarget; /* the window we are monitoring */
HWINEVENTHOOK g_hweh;
void CALLBACK WinEventProc(
    HWINEVENTHOOK hWinEventHook,
    DWORD         event,
    HWND          hwnd,
    LONG          idObject,
    LONG          idChild,
    DWORD         idEventThread,
    DWORD         dwmsEventTime)
{
 if (event == EVENT_OBJECT_DESTROY &&
     hwnd == g_hwndTarget &&
     idObject == OBJID_WINDOW &&
     idChild == INDEXID_CONTAINER) {
  PostMessage(g_hwnd, WM_CLOSE, 0, 0);
 }
}
```

The `WinEventHook` function is where it all happens. If our callback is told that a window was destroyed, and the window handle matches the one we are monitoring, then post ourselves a `WM_CLOSE` message, which will close the window and exit the program.

The rest is just scaffolding to get to the point where our `WinEventHook` gets called.

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 DWORD dwProcessId;
 DWORD dwThreadId = GetWindowThreadProcessId(g_hwndTarget,
                                            &dwProcessId);
 if (dwThreadId)
 g_hweh = SetWinEventHook(
     EVENT_OBJECT_DESTROY, EVENT_OBJECT_DESTROY,
     NULL, WinEventProc,
     dwProcessId, dwThreadId, WINEVENT_OUTOFCONTEXT);
 return g_hweh != NULL;
}
```

To register the hook, we obtain the thread ID and process ID of the window we are interested in tracking, then use the `SetWinEventHook` function to register our callback function, saying that we want to receive only `EVENT_OBJECT_DESTROY` notifications by passing it as both the `eventMin` and `eventMax`. We give it our callback function, and since we ask for `WINEVENT_OUTOFCONTEXT`, we don’t need to pass a module handle since we are not requesting injection.

Notice that we restrict our hook as much as we can. We specify that we care only about one event, and we are interested in only one process and only one thread. It's generally a good idea to restrict the hook as much as possible.

Of course, we also have to unregister the hook when we're done.

```
void
OnDestroy(HWND hwnd)
{
 if (g_hweh) UnhookWinEvent(g_hweh);
 PostQuitMessage(0);
}
```

And finally, we use our command line to specify the title of the window we are monitoring.

```
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
 ...
  g_hwndTarget = FindWindowA(lpCmdLine);
  g_hwnd =
  hwnd = CreateWindow(
 ...
}
```

With the Run dialog open, run this program with the command line argument `Run`. The program window opens, and when you click *Cancel* in the Run dialog, the program window closes. Wow that was exciting.

Bonus chatter: Remember that the window manager needs a message pump in order to call you back unexpectedly.

Exercise: Since we registered for only one thing, why did we have to perform the tests in `WinEventProc`? Why not just simplify the function to this?

```
void CALLBACK WinEventProc(
    HWINEVENTHOOK hWinEventHook,
    DWORD         event,
    HWND          hwnd,
    LONG          idObject,
    LONG          idChild,
    DWORD         idEventThread,
    DWORD         dwmsEventTime)
{
 PostMessage(g_hwnd, WM_CLOSE, 0, 0);
}
```

Exercise: With the Run dialog open, run this program with the command line argument `Run`. Now instead of clicking *Cancel* in the Run dialog, type some garbage into the edit control and then click OK. The Run dialog goes away and an error message appears instead. Why is the scratch program still running?

Raymond Chen

Follow