Using accessibility to monitor windows as they come and go

devblogs.microsoft.com/oldnewthing/20130325-00

March 25, 2013

Raymond Chen

Today's Little Program monitors windows as they come and go. When people contemplate doing this, they come up with ideas like installing a `WH_CBT` hook or a `WH_SHELL` hook, but one of the major problems with those types of hooks is that they are injected hooks. Injection is bad for a number of reasons.

- It forces the hook to be in a DLL so it can be injected.
- Hook activities need to be marshaled back to the main program.
- Your DLL will capture events only in processes of the same bitness, because you cannot load a 32-bit DLL into a 64-bit process or vice versa.
- You can inject into an elevated process only if your process is also elevated. If your process is non-elevated, then you will not capture events for windows belonging to elevated processes.

This is where accessibility comes in handy, because accessibility lets you specify whether you want your hook to be an injected or non-injected one. And if you're non-injected, then the programming model is much simpler because everything happens in your process (indeed, on a single thread).

Take the scratch program and make the following changes:

```
#include <strsafe.h>


BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 g_hwndChild = CreateWindow(TEXT(“listbox”), NULL,
     LBS_HASSTRINGS | WS_CHILD | WS_VISIBLE | WS_VSCROLL,
     0, 0, 0, 0, hwnd, NULL, g_hinst, 0);
 if (!g_hwndChild) return FALSE;
 return TRUE;
}


void CALLBACK WinEventProc(
    HWINEVENTHOOK hWinEventHook,
    DWORD event,
    HWND hwnd,
    LONG idObject,
    LONG idChild,
    DWORD dwEventThread,
    DWORD dwmsEventTime
)
{
 if (hwnd &&
     idObject == OBJID_WINDOW &&
     idChild == CHILDID_SELF)
 {
  PCTSTR pszAction = NULL;
  TCHAR szBuf[80];
  switch (event) {
  case EVENT_OBJECT_CREATE:
   pszAction = TEXT(“created”);
   break;
  case EVENT_OBJECT_DESTROY:
   pszAction = TEXT(“destroyed”);
   break;
  }
  if (pszAction) {
   TCHAR szClass[80];
   TCHAR szName[80];
   szClass[0] = TEXT(‘\0’);
   szName[0] = TEXT(‘\0’);
   if (IsWindow(hwnd)) {
    GetClassName(hwnd, szClass, ARRAYSIZE(szClass));
    GetWindowText(hwnd, szName, ARRAYSIZE(szName));
   }
   TCHAR szBuf[80];
   StringCchPrintf(szBuf, ARRAYSIZE(szBuf),
                    TEXT(“%p %s \”%s\” (%s)”), hwnd, pszAction,
                    szName, szClass);
   ListBox_AddString(g_hwndChild, szBuf);
  }
 }
}


int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
 …
  ShowWindow(hwnd, nShowCmd);


 HWINEVENTHOOK hWinEventHook = SetWinEventHook(
     EVENT_OBJECT_CREATE, EVENT_OBJECT_DESTROY,
     NULL, WinEventProc, 0, 0,
     WINEVENT_OUTOFCONTEXT | WINEVENT_SKIPOWNPROCESS);


  while (GetMessage(&msg, NULL, 0, 0)) {
   TranslateMessage(&msg);
   DispatchMessage(&msg);
  }


  if (hWinEventHook) UnhookWinEvent(hWinEventHook);
…
}
```

This is a generalization of our earlier program which waits for a specific window to be destroyed, except that we now are watching *all* windows for creation and destruction.

When you run this program, you see that there is a lot of window activity, but maybe you are interested only in windows when they are shown and hidden. No problem, that's a small change:

```
  switch (event) {
  case EVENT_OBJECT_SHOW:
   pszAction = TEXT(“shown”);
   break;
  case EVENT_OBJECT_HIDE:
   pszAction = TEXT(“hidden”);
   break;
  }
…


 HWINEVENTHOOK hWinEventHook = SetWinEventHook(
     EVENT_OBJECT_SHOW, EVENT_OBJECT_HIDE,
     NULL, WinEventProc, 0, 0,
     WINEVENT_OUTOFCONTEXT | WINEVENT_SKIPOWNPROCESS);
```

Notice that these notifications are received for windows from both 32-bit and 64-bit processes, and that they are received even for windows belonging to elevated processes. You can’t do that with an injected hook.

Raymond Chen

Follow