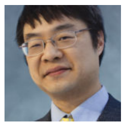
Reading mouse input from a console program, and programmatically turning off Quick Edit mode

devblogs.microsoft.com/oldnewthing/20130506-00

May 6, 2013



Raymond Chen

Today's Little program shows how to read mouse input from a console program. You might use this if you are writing a console-mode text editor with mouse support, or maybe you want to want to add mouse support to your roguelike game.

But I'm not going to implement the game itself. Instead, I'm just going to print mouse coordinates to the screen.

```
#define UNICODE
#define _UNICODE
#include <windows.h>
#include <tchar.h>
#include <stdio.h>


int __cdecl _tmain(int argc, PTSTR argv[])
{
 HANDLE hConsole = GetStdHandle(STD_INPUT_HANDLE);
 BOOL fContinue = TRUE;
 DWORD dwEvents;
 INPUT_RECORD input;
 while (fContinue &&
        ReadConsoleInput(hConsole, &input, 1, &dwEvents) &&
        dwEvents > 0) {
  switch (input.EventType) {
  case KEY_EVENT:
   if (input.Event.KeyEvent.wVirtualKeyCode == VK_ESCAPE) {
    fContinue = FALSE;
   }
  case MOUSE_EVENT:
   _tprintf(TEXT("X=%d, Y=%d; buttons=%x, shift=%x, flags=%x\n"),
     input.Event.MouseEvent.dwMousePosition.X,
     input.Event.MouseEvent.dwMousePosition.Y,
     input.Event.MouseEvent.dwButtonState,
     input.Event.MouseEvent.dwControlKeyState,
     input.Event.MouseEvent.dwEventFlags);
   break;
  }
 }
 return 0;
}
```

Our program just loops around collecting input, and if it gets a mouse event, it just prints the coordinates. “Insert game here.” If the user presses the `Esc` key, then we exit.

Run this program, move the mouse around the window, and… hey, nothing happened!

Oh right, because we forgot to enable mouse input. Let’s try that again.

```
 …
 HANDLE hConsole = GetStdHandle(STD_INPUT_HANDLE);


 DWORD dwPreviousMode = 0;
 GetConsoleMode(hConsole, &dwPreviousMode);
 DWORD dwNewMode = dwPreviousMode | ENABLE_MOUSE_INPUT;
 SetConsoleMode(hConsole, dwNewMode);
 }


 BOOL fContinue = TRUE;
 …
 }


 SetConsoleMode(hConsole, dwPreviousMode);


 return 0;
}
```

Remember, this is just a Little Program, so there is little to no error checking.

Okay, now you can run the program, and as you move the mouse around the window, you get… Well, it depends. Some of you may get output, and others may get nothing.

Those of you who got nothing aren't getting anything because you set *Quick Edit* mode on your console. *Quick Edit* mode commandeers the mouse and uses it for copy/paste operations rather than passing it through to the application. It's handy if you spend most of your time using programs that don't use the mouse, since it saves you from having to go to the *Edit* menu all the time.

It's not so handy if you're running a program that actually wants to use the mouse.

Add another line of code to the program to disable Quick-Edit mode:

```
 DWORD dwNewMode = dwPreviousMode | ENABLE_MOUSE_INPUT;
 dwNewMode &= ~ENABLE_QUICK_EDIT_MODE;
 SetConsoleMode(hConsole, dwNewMode);
```

Okay, now when you run the program and move the mouse around, you get… still nothing.

Ah, because `ENABLE_QUICK_EDIT_MODE` is an extended flag, and if you want to modify an extended flag, you also have to pass the `ENABLE_EXTENDED_FLAGS` flag. (You can guess how I discovered this.)

```
dwNewMode &= ~ENABLE_QUICK_EDIT_MODE;
SetConsoleMode(hConsole, dwNewMode |
                         ENABLE_EXTENDED_FLAGS);


…
SetConsoleMode(hConsole, dwPreviousMode |
                         ENABLE_EXTENDED_FLAGS);
```

Okay, now you can run the program, and as you wave the mouse around, the coordinates will be printed to the screen. Whew.

Exercise: Discuss why there is the crazy `ENABLE_EXTENDED_FLAGS` flag. For bonus points, come up with a way the flag could have been avoided while still solving the problem the flag was created for.

Raymond Chen

Follow