If I have multiple attached keyboards, how can I read input from each one individually?

devblogs.microsoft.com/oldnewthing/20160627-00

June 27, 2016

Raymond Chen

Raw Input is a feature of Windows that lets you obtain keyboard, mouse, or generic HID input. Okay, the generic HID input is nice, but the thing that is interesting today is the fact that the keyboard and mouse input is tagged with the device that generated it. This means that if you have multiple keyboards connected to your computer (say, the laptop integrated keyboard plus an external USB keyboard), you can distinguish the two input sources.

Let's do it.

Remember that Little Programs do very little to no error checking.

As usual, start with the scratch program and make these change:

```
#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);

  RAWINPUTDEVICE dev;
  dev.usUsagePage = 1;
  dev.usUsage = 6;
  dev.dwFlags = 0;
  dev.hwndTarget = hwnd;
  RegisterRawInputDevices(&dev, 1, sizeof(dev));

  return TRUE;
}

void
OnDestroy(HWND hwnd)
{
  RAWINPUTDEVICE dev;
  dev.usUsagePage = 1;
  dev.usUsage = 6;
  dev.dwFlags = RIDEV_REMOVE;
  dev.hwndTarget = hwnd;
  RegisterRawInputDevices(&dev, 1, sizeof(dev));

  PostQuitMessage(0);
}
```

First, we create a list box which we will use to display the input we receive.

Next, we register our window to receive raw keyboard input. The magic numbers for keyboard are Usage Page 1 and Usage 6. These magic numbers come from the USB HID specification.

The flip side of the coin is that we unregister when our window is destroyed.

Now the fun part: Receiving the input!

```
#define HANDLE_WM_INPUT(hwnd, wParam, lParam, fn) \
  ((fn)((hwnd), GET_RAWINPUT_CODE_WPARAM(wParam), \
        (HRAWINPUT)(lParam)), 0)

void OnInput(HWND hwnd, WPARAM code, HRAWINPUT hRawInput)
{
  UINT dwSize;
  GetRawInputData(hRawInput, RID_INPUT, nullptr,
                  &dwSize, sizeof(RAWINPUTHEADER));
  RAWINPUT *input = (RAWINPUT *)malloc(dwSize);
  GetRawInputData(hRawInput, RID_INPUT, input,
                  &dwSize, sizeof(RAWINPUTHEADER));
  if (input->header.dwType == RIM_TYPEKEYBOARD) {
    TCHAR prefix[80];
    prefix[0] = TEXT('\0');
    if (input->data.keyboard.Flags & RI_KEY_E0) {
        StringCchCat(prefix, ARRAYSIZE(prefix), TEXT("E0 "));
    }
    if (input->data.keyboard.Flags & RI_KEY_E1) {
        StringCchCat(prefix, ARRAYSIZE(prefix), TEXT("E1 "));
    }

    TCHAR buffer[256];
    StringCchPrintf(buffer, ARRAYSIZE(buffer),
        TEXT("%p, msg=%04x, vk=%04x, scanCode=%s%02x, %s"),
        input->header.hDevice,
        input->data.keyboard.Message,
        input->data.keyboard.VKey,
        prefix,
        input->data.keyboard.MakeCode,
        (input->data.keyboard.Flags & RI_KEY_BREAK)
            ? TEXT("release") : TEXT("press"));
    ListBox_AddString(g_hwndChild, buffer);
  }
  DefRawInputProc(&input, 1, sizeof(RAWINPUTHEADER));
  free(input);
}

...
    HANDLE_MSG(hwnd, WM_INPUT, OnInput);
```

When we get the `WM_INPUT` message, we use the `GetRawInputData` function to convert the raw input handle to a raw input structure. This involves the standard two-step of first finding out how much memory you need, then allocating that memory and trying again.

Do note that if you are going to use a preallocated buffer (for example, to handle the common case where the raw input fits in less than 80 bytes), your buffer still must be properly aligned for a `RAWINPUT` structure. This is one of the basic ground rules, but it's worth calling out

explicitly because you are going to be tempted to preallocate the buffer. We didn't have to worry about it here because the `malloc` function guarantees that the allocated buffer is suitably aligned.

Next, we confirm that the input is keyboard input. This is theoretically not necessary because the only input we registered for is keyboard input, but I feel better checking for it, because somebody might do a `RegisterRawInputDevices` and register some other type of input, and I don't want to get faked out.

After verifying that we do indeed have keyboard input, we extract the payload:

- The device handle tells us which keyboard generated the input.
- The Message is the window message that was generated.
- The VKey is the virtual key code.
- The MakeCode is the scan code.
- The Flags provide other information:
  - Which prefixes are present on the scan code.
  - Whether this is a make (press) or break (release).

Finally, we call `DefRawInputProc` to allow default processing to occur. This lets the keypress enter the normal input system.

Note that although there's a `GetRawInputDeviceList` function which lets you find all the keyboard devices, that is not useful in practice because modern computers have a ton of special-purpose keyboards hiding inside them. For example, the volume control knobs on your laptop might actually be a tiny two-button keyboard.

Raymond Chen

Follow