How can I detect whether the system has a keyboard attached? On the GetRawInputDeviceList function

devblogs.microsoft.com/oldnewthing/20220302-00

March 2, 2022

Raymond Chen

Last time, we saw that the instructions for unlocking the PC vary depending on whether a keyboard is attached. So how do you detect whether a keyboard is attached?

The `GetRawInputDeviceList` function gives you the raw input devices that are registered with the input system. Let's write a little program to count how many of them are keyboards.

```
#include <windows.h>
#include <stdio.h> // Horrors! Mixing C and C++!
#include <vector>

[[noreturn]] void throw_win32_error(DWORD error)
{
    // replace with your desired win32 exception object
    std::terminate();
}

std::vector<RAWINPUTDEVICELIST> GetRawInputDevices()
{
  UINT deviceCount = 0;
  if (GetRawInputDeviceList(nullptr, &deviceCount,
            sizeof(devices[0]) != 0) {
    throw_win32_error(GetLastError());
  }

  std::vector<RAWINPUTDEVICELIST> devices(deviceCount);
  while (deviceCount != 0) {
    UINT actualDeviceCount = GetRawInputDeviceList(
            devices.data(), &deviceCount,
            sizeof(devices[0]);
    if (actualDeviceCount != (UINT)-1) {
        devices.resize(actualDeviceCount);
        return devices;
    }
    DWORD error = GetLastError();
    if (error != ERROR_INSUFFICIENT_BUFFER) {
      throw_win32_error(error);
    }
    devices.resize(deviceCount);
  }
}
```

The behavior of the `GetRawInputDeviceList()` is very strange. Here it is in a table:

Scenario	**Buffer pointer**	**Size on entry**	**Size on return**	**Return value**
Query number of devices	`nullptr`	ignored	required size	0
Request data (failed)	Non-`nullptr`	provided size	required size	`0xFFFFFFFF`
Request data (succeeded)	Non-`nullptr`	provided size	unchanged	actual size

The `GetRawInputDevices()` function starts by using the *Query number of devices* pattern to get an initial guess as to the number of devices, and then goes into the usual loop of calling a function (in this case `GetRawInputDeviceList()`) to fill a buffer, resizing the buffer based on the result from the previous query, until you finally get what you want. There is a weird edge case where there are no input devices: A vector resized to zero is permitted to return `data() == nullptr`, but that would cause our *Request data* to be misinterpreted as a *Query number of devices*, so we skip the loop if the device count is zero.

Most people don't write the loop. They just call the function twice, once to get the count, and once to fill the buffer. But it means that if a device is attached between the two calls, the code fails because the count changed. You need to make it a loop so you can adapt to changes that occur behind your back.

Other common oversights when using the `GetRawInputDeviceList()` function are missing the case where there are no devices, or the case where a device is removed during the loop, so that the *Request data* succeeds but returns a value smaller than the provided size.

If we start with a nonzero initial guess, we can get rid of the preliminary portion of the function and go straight to the loop, thereby avoiding the weird "I'm going to return success even though you didn't get anything" first row of the above table.

```
std::vector<RAWINPUTDEVICELIST> GetRawInputDevices()
{
  UINT deviceCount = 10; // initial guess, must be nonzero
  std::vector<RAWINPUTDEVICELIST> devices(deviceCount);
  while (deviceCount != 0) {
    UINT actualDeviceCount = GetRawInputDeviceList(
            devices.data(), &deviceCount,
            sizeof(devices[0]);
    if (actualDeviceCount != (UINT)-1) {
        devices.resize(actualDeviceCount);
        return devices;
    }
    DWORD error = GetLastError();
    if (error != ERROR_INSUFFICIENT_BUFFER) {
      std::terminate(); // throw something
    }
    devices.resize(deviceCount);
  }
}
```

Now we get to use this function to study the raw input devices.

```
int main(int argc, char** argv)
{
  auto devices = GetRawInputDevices();
  int mouseCount = 0;
  int keyboardCount = 0;
  int otherCount = 0;
  for (auto const& device : devices) {
    switch (device.dwType)
    {
    case RIM_TYPEKEYBOARD: keyboardCount++; break;
    case RIM_TYPEMOUSE: mouseCount++; break;
    default: otherCount++; break;
    }
  }
  printf("There are %d keyboards, %d mice, and %d other things\n",
         keyboardCount, mouseCount, otherCount);
  return 0;
}
```

We walk through the list and tally up how many devices there are of each kind.

If you just want a "yes or no" answer about whether there is a keyboard attached, you can ask `KeyboardCapabilities.KeyboardPresent`:

```
#include <stdio.h> // Horrors! Mixing C and C++!
#include <winrt/Windows.Devices.Input.h>

int main(int argc, char** argv)
{
  winrt::init_apartment();
  winrt::Windows::Devices::Input::KeyboardCapabilities capabilities;
  printf("KeyboardPresent = %d\n", capabilities.KeyboardPresent());
  return 0;
}
```

The `KeyboardPresent` property identifies keyboards by… enumerating the raw input devices and seeing if any of them is a keyboard.

This all sounds good, except that a lot of devices report themselves as keyboards, even though they aren't keyboards in the usual sense. Next time, we'll see if we can filter those guys out.

Raymond Chen

Follow