# USBHID
## Programmer's Interface

Version 1.6
March 2024

The usbhid library is a programmer's interface to the range of pushbuttons and switch input boards by CompuPhase. These devices have a "HID" interface, for which every modern operating system has built-in drivers.

This library is available in the form of 32-bit and 64-bit DLLs for Microsoft Windows, and a "shared object" for Linux (64-bit only). The library also includes interface files for various programming languages.

For downloads, see: https://www.compuphase.com/downloads/

# Contents

## Trademarks

“CompuPhase” is a trademark of CompuPhase.

“Linux” is a registered trademark of Linus Torvalds.

“Microsoft” and “Microsoft Windows” are registered trademarks of Microsoft Corporation.

“Unicode” is a registered trademark of Unicode, Inc.

# Copyright

ⓒ Copyright 2018–2024, CompuPhase; Eerste Industriestraat 19 Bussum, The Netherlands
voice: (+31)-(0)35 6939 261; fax: (+31)-(0)35 6939 293
e-mail: info@compuphase.com; homepage: http://www.compuphase.com

The examples and programs in this manual have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose.

# Using usbhid

The first function to call is `usbhid_Initialize`. It collects information on all supported devices that are connected to the workstation. It returns the number of devices that it has detected.

Most other functions take a device index, between zero and the number of devices (the number returned by `usbhid_Initialize`). The devices can be uniquely identified by their model and serial numbers. You can retrieve these numbers for each device using `usbhid_GetInfo`. Thus, you can map a device index to a specific device.

A wireless dongle supports up to six (wireless) buttons. `usbhid_GetInfo` returns the serial number of the device inserted in the USB port —meaning: the dongle. To get the serial numbers of the buttons connected to the dongle, use `usbhid_GetSwitchSerial`.

For buttons, the alternative is to identify each device by the key code sequences defined in each button. This assumes, of course, that all buttons that are connected to a workstation are each configured to transmit a different key code (or key sequence). Function `usbhid_GetKeyString` serves this purpose.

The principal use for usbhid is to switch the LED illumination on or off for illuminated buttons, or to set or release a relay on a relay driver board. These are the functions `usbhid_SetLED` and `usbhid_SetOutput`.

When the buttons are used as "buzzers" in a quiz, you will likely need a "lockout" system that disables the other buttons immediately after a first button is pressed. This can be handled in multiple ways. This API provides the `usbhid_SetDisabled` function to disable (and re-enable) buttons.

## Examples

Below is a complete C program that uses an assortment of functions from the usbhid library. It must be linked with the appropriate library; 32-bit and 64-bit import libraries are provided for Microsoft Windows. Linux programs can link directly against the shared library (for Linux, the call to the `Sleep` function must also be replaced with `usleep` in the example below).

LISTING: C/C++ program example

```
#include <stdio.h>
#include "usbhid.h"

int main(void)
{
  unsigned count, idx, iter;

  /* initialize the library */
  count = usbhid_Initialize();
  printf("Devices found: %d\n", count);
```

```
  /* show connected devices */
  for (idx = 0; idx < count; idx++) {
    unsigned serial;
    int model;
    char keystring[100];
    usbhid_GetInfo(idx, &serial, &model);
    usbhid_GetKeyString(idx, 0, keystring, 100);
    printf("Device %d: model %d, serial %u, key \"%s\"\n",
           idx, model, serial, keystring);
  }

  /* find the first device that is an illuminated button */
  for (idx = 0; idx < count; idx++) {
    int model;
    usbhid_GetInfo(idx, NULL, &model);
    if (model == 710 || model == 720)
      break;
  }

  /* blink the LED a few times on that button */
  for (iter = 0; iter < 10; iter++) {
    usbhid_SetLED(idx, (iter & 1) ? LED_OFF : LED_ON, 0);
    Sleep(500);
  }

  usbhid_Close();
  return 0;
}
```

A C# example for Microsoft Windows is below. A C# project must be build with the "`usbhid.cs`" interface file. This file selects between the 32-bit and 64-bit DLLs as appropriate.

The C# and VB.Net interface files declare all DLL functions in a wrapper class called `usbhid`. Therefore, where a C/C$^{++}$ program would call `usbhid_Initialize()`, a C# or VB.Net program would invoke this function as `usbhid.Initialize()` (a dot instead of an underscore).

LISTING: C# program example

```
using System;
using System.Text;
using USBHID;

namespace DemoUSBHID {
  class Demo {
    static void Main() {

      /* initialize the library */
      uint count = usbhid.Initialize();
      Console.WriteLine("Devices found: {0}", count);

      /* show connected devices */
      uint idx;
      for ( idx = 0; idx < count; idx++ ) {
        uint serial = 0;
        int model = 0;
        StringBuilder keystring = new StringBuilder(100);
        usbhid.GetInfo(idx, ref serial, ref model);
        usbhid.GetKeyString(idx, 0, keystring, 100);
```

```
        Console.WriteLine("Device {0}: model {1}, serial {2}, key \"{3}\"",
                          idx, model, serial, keystring);
      }
      /* find the first device that is an illuminated button */
      for ( idx = 0; idx < count; idx++ ) {
        uint serial = 0;
        int model = 0;
        usbhid.GetInfo(idx, ref serial, ref model);
        if ( model == 710 || model == 720 ) break;
      }
      /* blink the LED a few times on that button */
      for (uint iter = 0; iter < 10; iter++){
        usbhid.SetLED(idx, (iter & 1) == 1 ? LEDFunction.OFF : LEDFunction.ON, 0);
        System.Threading.Thread.Sleep(500);
      }
      usbhid.Close();
    }
  }
}
```

# Function reference

## usbhid_Close — Frees all allocated resources

`usbhid_Close` frees memory and resources that are allocated during the session with the library.

**C/C++:** `void usbhid_Close()`

**C#:** `void Close()`

**VB.Net:** `Sub Close()`

Returns: —

See also: `usbhid_Initialize`

## usbhid_GetInfo — Return information on a device

`usbhid_GetInfo` returns the model and serial numbers of a connected device, by which it can be uniquely identified.

**C/C++:**
```
BOOL usbhid_GetInfo(unsigned DeviceIndex,
                    unsigned *SerialNumber,
                    int *Model)
```

**C#:**
```
bool GetInfo(uint DeviceIndex,
             ref uint SerialNumber,
             ref int Model)
```

**VB.Net:**
```
Function GetInfo(ByVal DeviceIndex As UInteger,
                 ByRef SerialNumber As UInteger,
                 ByRef Model As Integer) As Boolean
```

`DeviceIndex`
: The index of the device, a value between 0 and the number of devices.

`SerialNumber`
: Holds the serial number of the device upon return.

`Model`
: Holds the model number of the device upon return.

Returns: In C: `TRUE` on success, `FALSE` on failure.
In C# & C++: `true` on success, `false` on failure.

Notes: Current model numbers are:

| | |
|---|---|
| 700 | large dome button (non-illuminated) |
| 710 | illuminated large dome button |
| 720 | illuminated pushbutton |
| 730 | wireless dongle (for the battery-powered buttons) |
| 735 | wireless dongle for the battery-free wireless buttons |
| 740 | pendant switch |

See also: usbhid_GetKeyString, usbhid_GetSwitchSerial, usbhid_Initialize

---

## usbhid_GetKeyString — Return the button's key code sequence

usbhid_GetInfo returns the key code (or the key sequence) for a button device. This key code sequence may be used to identify the button (when multiple buttons are connected to a workstation, each is typically configured to a different key combination).

**C/C++:**
```
BOOL usbhid_GetKeyString(unsigned DeviceIndex,
                         unsigned SwitchIndex,
                         TCHAR *KeyString,
                         unsigned MaxLength)
```

**C#:**
```
bool GetKeyString(uint DeviceIndex,
                  uint SwitchIndex,
                  StringBuilder KeyString,
                  uint MaxLength)
```

**VB.Net:**
```
Function GetKeyString(ByVal DeviceIndex As UInteger,
                      ByVal SwitchIndex As UInteger,
                      ByVal KeyString As StringBuilder,
                      ByVal MaxLength As UInteger)
                      As Boolean
```

`DeviceIndex`
: The index of the device, a value between 0 and the number of devices.

`SwitchIndex`
: For devices that support multiple switches (such as the wireless dongles), this parameter indicates the switch index, starting from 0. For devices that support a single switch, this parameter should be set to zero.

`KeyString`
: Holds a description of the key code (or the key sequence). See the notes for details.

`MaxLength`
: The size in characters of parameter `KeyString`.

Returns: In C: `TRUE` on success, `FALSE` on failure.
In C# & C++: `true` on success, `false` on failure.
This function fails when the device is not a button.

Notes: In Microsoft Windows, this function is available in ASCII and Unicode variants. In Linux, only the ASCII version is available.

When a button is configured to transmit a single digit or alphabetical character, the return value is a string with that digit or letter as its single element. Keys that are not digits or letters are returned by a descriptive label, for example "`Enter`".

A key may be prefixed with shift codes. For example, "`Shift + Tab`" means that the TAB key was typed with the shift key pressed. Other shift codes are "`Alt`", "`Ctrl`" and "`GUI`". The "GUI" key is the "Windows" key in Microsoft Windows, and the "Apple" key (or "command" key) in Apple OSX.

See also: `usbhid_GetInfo`, `usbhid_Initialize`, `usbhid_SetLayout`

---

## usbhid_GetSwitchSerial — Get the serial number of a switch

`usbhid_GetSwitchSerial` returns the serial number of a switch that is attached to a dongle.

**C/C++:**
```
BOOL usbhid_GetSwitchSerial(unsigned DeviceIndex,
                            unsigned SwitchIndex,
                            unsigned *SerialNumber)
```

**C#:**
```
bool GetSwitchSerial(uint DeviceIndex,
                     uint SwitchIndex,
                     ref uint SerialNumber)
```

**VB.Net:**
```
Function GetSwitchSerial(ByVal DeviceIndex As UInteger,
                         ByVal SwitchIndex As UInteger,
                         ByRef SerialNumber As UInteger)
                         As Boolean
```

`DeviceIndex`
: The index of the device, a value between 0 and the number of devices.

`SwitchIndex`
: For devices that support multiple switches (such as the wireless dongles), this parameter indicates the switch index, starting from 0. For devices that support a single switch, this parameter should be set to zero.

`SerialNumber`
: Holds the serial number of the device upon return.

Returns: In C: `TRUE` on success, `FALSE` on failure.
In C# & C++: `true` on success, `false` on failure.
This function fails when the device is not a button or switch.

Notes: For devices that support multiple switches (such as the wireless dongles), function `usbhid_GetInfo` returns the serial number of the dongle. This function returns the serial number of the connected switches.

For devices that support only a single switch, this function succeeds only when `SwitchIndex` is zero, and the serial number that it returns is the same as the one returned by `usbhid_GetInfo`.

See also: `usbhid_GetInfo`

---

## usbhid_Initialize — Initialize the library

`usbhid_Initialize` initializes the library and returns the number of supported devices.

**C/C++:**
```
unsigned usbhid_Initialize()
```

**C#:**
```
uint Initialize()
```

**VB.Net:**
```
Function Initialize() As UInteger
```

Returns: The number of supported devices connected to the workstation, or 0 on failure.

Notes: This function must be called before any other function from the usbhid library. The memory and resources allocated by this function are freed with `usbhid_Close`.

After a device is connected or disconnected, this function should be called again, so that the device list is updated.

See also: `usbhid_Close`, `usbhid_GetInfo`

---

## usbhid_IsSwitchDown — Test switch status

`usbhid_Initialize` checks whether a switch is “down”.

**C/C++:**
```
BOOL usbhid_IsSwitchDown(unsigned DeviceIndex,
                         unsigned SwitchIndex)
```

**C#:**
```
bool IsSwitchDown(uint DeviceIndex, uint SwitchIndex)
```

**VB.Net:**
```
IsSwitchDown(ByVal DeviceIndex As UInteger,
             ByVal SwitchIndex As UInteger) As Boolean
```

`DeviceIndex`
: The index of the device, a value between 0 and the number of devices.

`SwitchIndex`
: For devices that support multiple switches (such as the wireless dongles), this parameter indicates the switch index, starting from 0. For devices that support a single switch, this parameter should be set to zero.

Returns: In C: `TRUE` on success, `FALSE` on failure.
In C# & C++: `true` on success, `false` on failure.

Notes: One use for this function is to check whether a "key input" event came from a switch or from the keyboard. Alternatively, for a system without display or terminal, this function may be needed to capture switch input (your application may not get "focus" when the operating system fails to detect a display).

The USB HID device latches the "switch down" state for 0.2 seconds. So when a switch is released within 0.2 seconds after being pressed, the returned status will still be reported as"down" for 0.2 seconds.

As a result of the above, if an application relies on this function to capture switch input, rather than responding to "key input" events, the application should poll this function at an interval below 0.2 seconds.

See also: `usbhid_GetInfo`, `usbhid_SetLed`

---

## usbhid_SetDisabled

Temporarily disables a switch

`usbhid_SetDisabled` disables a switch until it is re-enabled or until it is power-cycled. When disabled, a switch does not transmit button presses or releases (but it still responds to other commands, such as switching the illumination on or off).

**C/C++:**
```
BOOL usbhid_SetDisabled(unsigned DeviceIndex,
                        unsigned short Disabled)
```

**C#:**
```
bool SetDisabled(uint DeviceIndex,
                 ushort Disabled)
```

**VB.Net:**
```
Function SetDisabled(ByVal DeviceIndex As UInteger,
                     ByVal Disabled As UShort) As Boolean
```

`DeviceIndex`
: The index of the device, a value between 0 and the number of devices.

`Disabled`
: Should be 1 to disable the switch, or 0 to re-enabled it.

Returns: In C: `TRUE` on success, `FALSE` on failure.
In C# & C++: `true` on success, `false` on failure.

Notes: When a switch is disabled while it is pressed, the switch will immediately transmit a "key-up" status to the workstation, but ignore any further presses or releases of the switch. Function `usbhid_IsSwitchDown` returns *false* for a disabled switch, regardless of its true state. When a switch is enabled while it was pressed, it will transmit a "key-down" status to the workstation immediately after being enabled.

See also: `usbhid_IsSwitchDown`, `usbhid_SetLED`

---

## usbhid_SetLayout — Set keyboard layout

`usbhid_SetLayout` sets the keyboard layout, to use for `usbhid_GetKeyString`.

**C/C++:** `BOOL usbhid_SetLayout(int Layout)`

**C#:** `bool SetLayout(int Layout)`

**VB.Net:** `Function SetLayout(ByVal Layout As Integer) As Boolean`

`Layout`
: Can be one of the following:

| | |
|---|---|
| `LAYOUT_BELGIAN` | AZERTY |
| `LAYOUT_CANADIAN` | |
| `LAYOUT_DANISH` | |
| `LAYOUT_FRENCH` | AZERTY |
| `LAYOUT_GERMAN` | QWERTZ |
| `LAYOUT_INTERNATIONAL` | "*US International*" layout |
| `LAYOUT_ITALIAN` | |
| `LAYOUT_LATIN_AMERICAN` | |
| `LAYOUT_NORWEGIAN` | |
| `LAYOUT_PORTUGUESE` | |
| `LAYOUT_SPANISH` | |
| `LAYOUT_SWEDISH` | |
| `LAYOUT_UK` | UK "*Extended*" layout |
| `LAYOUT_US` | |

For C#, see also the notes.

Returns: In C: `TRUE` on success, `FALSE` on failure.
In C# & C++: `true` on success, `false` on failure.

Notes: The pushbuttons (and other input devices) store the key code(s) in a universal encoding as defined in the USB standard. The actual function ofa key, depends on the active "layout" in the operating system. As a result, the string returned by `usbhid_GetKeyString` is also interpreted to the active layout. However, this usbhid library requires the application to set the layout to use.

The default layout is "US".

In C#, the values for parameter `Layout` are in the enumerated type `KbdLayout`.

See also: `usbhid_GetKeyString`

---

## usbhid_SetLED — Turn LED illumination on or off

`usbhid_SetLED` switches the illumination on or off, on illuminated buttons. This function has no effect on non-illuminated buttons.

**C/C++:**
```
BOOL usbhid_SetLED(unsigned DeviceIndex,
                   unsigned short Function,
                   unsigned short Dimming)
```

**C#:**
```
bool SetLED(uint DeviceIndex,
            LEDFunction Function,
            ushort Dimming)
```

**VB.Net:**
```
Function SetLED(ByVal DeviceIndex As UInteger,
                ByVal LEDFunc As LEDFunction,
                ByVal Dimming As UShort) As Boolean
```

`DeviceIndex`
: The index of the device, a value between 0 and the number of devices.

`Function`
: Can be one of the following:

| | |
|---|---|
| `LED_OFF` | Turn illumination off. |
| `LED_ON` | Turn illumination on (continuously). |
| `LED_BLINK` | Set the illumination to blink. |

For C#, see also the notes.

`Dimming`
: The dimming level for illumination, see the notes.

Returns: In C: `TRUE` on success, `FALSE` on failure.
In C# & C++: `true` on success, `false` on failure.

Notes: On models that support dimming, parameter `Dimming` holds a value between 0 and 15, where 0 is full intensity and 15 is the lowest illumination level. On models that do not support dimming, parameter `Dimming` is ignored.

In C#, the values for parameter `Function` are in the enumerated type `LEDFunction`, which has elements `OFF`, `ON` and `BLINK`.

See also: `usbhid_Initialize`, `usbhid_SetOutput`

---

## usbhid_Version — Get the version of the library

`usbhid_Version` returns the version of the library.

**C/C++:** `unsigned usbhid_Version()`

**C#:** `uint Version()`

**VB.Net:** `Function Version() As UInteger`

Returns: The version number of the usbhid library in a format where the minor version is in the low byte and the major version number is in the second byte. The return value `0x104` represents version 1.4.

# License

This manual and the software programs consisting of all the files included in the "usbhid" package and hereafter collectively referred to as "the product", is copyright © 2018–2024 CompuPhase. The product under this license is provided free of charge. You are granted a non-exclusive license to use the product for under the following conditions:

**YOU MAY**:

◇ Use the files in the product to build your own applications.

◇ Copy and distribute the product's DLL files with your applications, under a license of your own choice.

◇ Copy the product, and distribute copies of the product, provided that you distribute the complete and unmodified product (including this manual with the copyright, license and disclaimer of warranty).

**YOU MAY NOT**:

◇ Use the product for illegal purposes.

◇ Remove or conceal the copyright of the product, or claim copyright on (parts of) the product.

◇ Charge money or fees for redistributing the product, except to cover distribution costs.

**LIMITED WARRANTY**:

CompuPhase cannot be held liable for *any* damage or loss of profits that results from the use of the product (or part thereof), or from the inability to use it, to the extent that the law permits.

# Index