WCID Devices

Edit New Page Jump to bottom

jmcblitz edited this page on Feb 16, 2018 · 74 revisions

Table of Contents

- What is WCID?
- History
- Example
- WCID for end-users
- WCID for manufacturers
- Implementation
  - Microsoft OS String Descriptor
  - Microsoft Compatible ID Feature Descriptor
  - Windows Automated Driver Installation
  - Defining a Device Interface GUID or other device specific properties
  - Example of a complete OS Feature Descriptor query
- WCID Advantages
- WCID Drawbacks
- WCID Device Examples:
  - PIC
  - AVR (LUFA)
  - STM32
  - Other
- Aditional Links

What is WCID?

A **WCID device**, where WCID stands for "**W**indows **C**ompatible **ID**", is an USB device that provides extra information to a Windows system, in order to facilitate automated driver installation and, in most circumstances, allow immediate access.

WCID allows a device to be used by a Windows application almost as soon as it is plugged in, as opposed to the the usual scenario where an USB device that is neither HID nor Mass Storage requires end-users to perform a manual driver installation. As such, WCID can bring the 'Plug-and-Play' functionality of HID and Mass Storage to **any** USB device (that sports a WCID aware firmware).

WCID is an extension of the WinUSB Device functionality, put forward by Microsoft during the Windows 8 Developer Preview and that uses capabilities (Microsoft OS Descriptors, or MODs) that have been part of Windows since Windows XP SP2.

As of May 2012, an automated WinUSB WCID driver is provided on all platforms starting with Windows Vista. On Windows 8 or later, it is native to the system which means there is no need to go online for the driver to be installed, whereas, for Vista and Windows 7, it will be obtained from the internet, through Windows Update.

History

Before WCID, only USB devices that belonged to a well supported class such as HID (USB keyboards, mice, joysticks, etc.) or Mass Storage (USB disk, flash based storage, media players such as iPods, etc.) required no intervention for driver installation, since Windows took care of it automatically the first time the device was plugged in. On the other hand, other devices usually required users to manually provide and install the driver themselves. This detracted from ease of use and limited the the ability of people, who aren't familiar with the driver installation process on Windows, to use generic USB applications such as the ones based on libusb, LibUsbDotNet, libusb-win32, libusbK or WinUSB.

However, the methods for automated driver installation of HID or Mass Storage devices (reliance of a common class, instead of a specific device ID) and the method of providing specific OS related device information (Microsoft OS Descriptors), which have been present since Windows XP SP2, can actually be combined to allow for the automated driver installation of any type of USB device.

In 2011, WCID was put forward by Microsoft in the Windows 8 Developer Preview, through an updated WinUSB driver, and was confirmed to be applicable for more than just WinUSB devices.

In 2012, Microsoft also added the WinUSB WCID driver files to Windows Update, with the effect of bringing WCID capabilities and automated WinUSB driver installation to other recent versions of Windows.

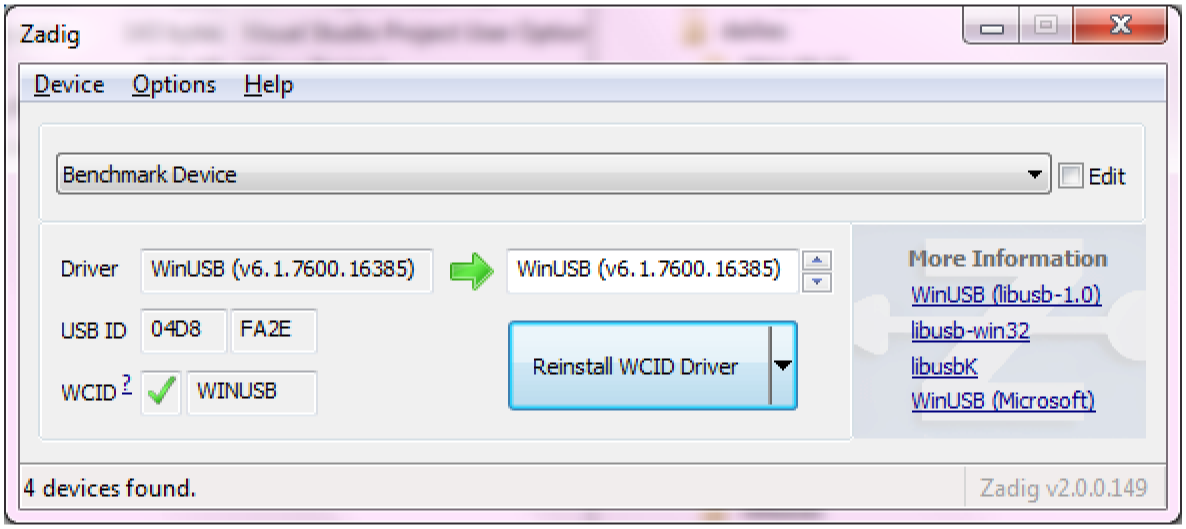
As it greatly enhances user experience, WCID is now being leveraged in USB devices such as Android devices, medical devices, game controllers, and various USB gadgets, with many more expected to follow. As a matter of fact, WCID is what Microsoft uses in their XBox 360 USB controller, to provided seamless installation of the XBox toolbar driver.

Example

The following is a screenshot of a "WINUSB" WCID device as detected by Zadig.

On Windows Vista or later, this device, which is neither HID nor Mass Storage, sees a driver installed automatically without any form of user intervention through either Windows Update (Windows Vista, Windows 7) or natively (Windows 8).

The same can also be achieved on Windows XP SP2, if a WinUSB WCID driver (i.e. a generic driver) has been installed at least once.



WCID for end-users

What WCID means for end-user is that, as long as a device is WCID, and a WCID driver with the same generic compatible ID is either available on the system or from Windows Update, no manual driver installation will be required when plugging the device. Instead, Windows will perform the driver installation automatically.

Moreover, an application such as Zadig can ensure that even on platforms where a WCID driver is not provided by Microsoft (eg. XP SP2) or when non Microsoft USB drivers are used (libusb0, libusbK), WCID can still be enabled through a one-time WCID driver installation so that, when you plug a WCID device in the future, you won't have anything to do.

Thus, if you are an end-user, and the device you want to use is neither HID or Mass Storage, you should ask the manufacturer whether it is WCID compatible. If not, you should ask them whether they can update the device firmware and convert it to WCID, which is very easy for them to do, as this means it will greatly enhance your user experience.

All in all, once manufacturers start using WCID, you should find that almost any USB device on Windows can actually become Plug-and-Play and that cumbersome USB driver installations will have become a thing of the past.

WCID for manufacturers

Obviously, the main advantage of WCID for device manufacturers is that it alleviates the need to provide your own drivers or your own custom driver installer application, as well as educate users with regards to driver requirements for devices that do not belong to well established USB classes. It also alleviates the need to go through WHQL, for every single device you produce, should your users want to obtain your driver through Windows Update.

For any user running Windows Vista or later, as long as you use `"WINUSB"` for your WCID type (in the Compatible ID Descriptor - see below), nothing more is required, and the WinUSB WCID driver, which either comes with the OS or can be fetched from Windows Update, will be installed automatically. Thus you can start developing WinUSB or libusb based applications, without having to worry about driver installation.

For end-users not running Windows Vista or later, or if you choose to use libusb-win32 or libusbK to develop your application, you can simply either provide a one-time WCID driver installer application, such as Zadig or libwdi's wdi-simple (both of which being Open Source and customizable to your needs), as these applications can install the WCID version of the WinUSB, libusb-win32 or libusbK drivers.

Also, while WCID requires the addition on additional descriptors, a lot of effort has been devoted in this page to ensure that you have all the information required to deploy WCID quickly and painlessly (i.e. near zero-cost), especially as the simple registry checks detailed below, the use of Zadig or the use of other test applications such as the libusb sample application xusb (a Windows binary version of which is also available here) or Linux's usb-utils (when updated to dump Microsoft OS Descriptors) can help validate whether your descriptors have been properly set.

For examples of actual WCID device implementations for AVR and PIC, as well as a typical WCID USB query trace, please have a look at the Examples section.

If positive customer experience is one of your priorities, and you want to reduce both development and support costs, you should seriously consider implementing WCID into your next device.

Implementation

You can find below an extended description of how WCID driver installation works on Windows, as well as the means you can use to confirm that your device is properly set for WCID.

Important: Only USB 2.0 devices or later can be recognized as WCID. As per the following:

> *If the USB Device Descriptor's bcdUSB field is equal to `0x0100` or `0x0110`, the hub driver will skip the query for the MS OS Descriptor and move to the "Serial Number String Descriptor Query" state*

Thus your Device Descriptor's bcdUSB field **must** be set to `0x0200` or higher.
Of course, this does not mean that your device has to operate at High Speed. It just means that your device complies with the USB 2.0 Specifications or higher.

Microsoft OS String Descriptor

The first time an USB device is plugged in, the Microsoft USB port driver (`usbport.sys`) will read the standard USB Descriptors, which includes the standard Device, Configuration, Interface and String Descriptors. It will also attempt to read an additional String Descriptor located at index `0xEE` . This String Descriptor, which is not mandated by the USB specifications, is a Microsoft extension called an OS String Descriptor and it is the first of two elements that establish whether a device is WCID.

When Windows checks for this String Descriptor, one of the first thing that happens is the creation of a registry entry, under `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\usbflags` , that is the concatenation of `VID+PID+BCD_RELEASE_NUMBER` (This key is never deleted). In this new section, a `osvc` 2 byte `REG_BINARY` key is created, that indicates whether an appropriate Microsoft OS String Descriptor was found.

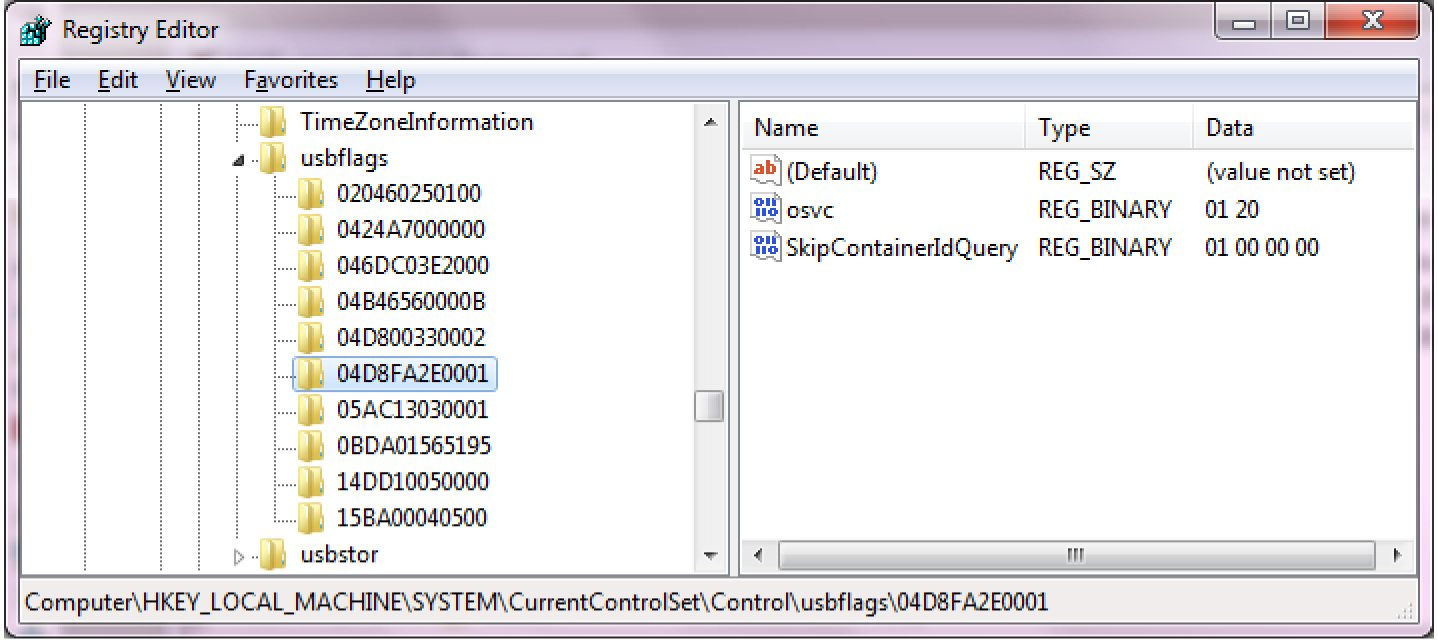
- If the `0xEE` descriptor doesn't exist, or if it doesn't match the Microsoft signature (`"MSFT100"`), then `osvc` is set to `0x0000` , which, in our case, means that the device is not WCID.
- If the `0xEE` descriptor exists and matches the expected organization of a Microsoft OS String Descriptor, then `oscv` is set to `0x01##` , where `01` indicates that the device satisfies the MS OS Vendor extensions and where `##` is the Vendor Code byte value (see below).

The following table details how exactly the Microsoft OS String Descriptor should be set in your firmware:

Table 1: Microsoft OS String Descriptor

Value	Type	Description
`0x12`	BYTE	Descriptor length (18 bytes)
`0x03`	BYTE	Descriptor type (3 = String)
`0x4D, 0x00, 0x53, 0x00, 0x46, 0x00, 0x54, 0x00, 0x31, 0x00, 0x30, 0x00, 0x30, 0x00`	7 WORDS Unicode String (LE)	Signature: `"MSFT100"`
`0x##`	BYTE	**Vendor Code**
`0x00`	BYTE	Padding

Below is an example of the key created by Windows 7 for a WCID device (LibusbDotNet's Benchmark device, with VID `0x04D8` , PID `0xFA2E` and BCD Release Number `0x0001`):



TimeZoneInformation
usbflags
020460250100
0424A7000000
046DC03E2000
04B46560000B
04D800330002
04D8FA2E0001
05AC13030001
0BDA01565195
14DD10050000
15BA00040500
usbstor

Name	Type	Data
(Default)	REG_SZ	(value not set)
osvc	REG_BINARY	01 20
SkipContainerIdQuery	REG_BINARY	01 00 00 00

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\usbflags\04D8FA2E0001

In the example above, the Vendor Code was set to `0x20` in the firmware. Thus, you can easily find out if Windows has been able to read a valid OS Vendor signature from your device by checking the registry.

Microsoft Compatible ID Feature Descriptor

If the osvc is valid, then additional Microsoft-defined Feature Descriptor are issued, with each call being broken down by the OS into header retrieval, to find the size, and then full payload retrieval (thus a device should see twice as many requests as calls that are issued).

At least one of these calls is for the `Compatible ID Feature Descriptor`

This is issued by the OS as a Vendor Request (control transfer), with `wIndex` set to `0x0004` and `bRequest` set to the Vendor Code previously returned, as well as the recipient part of `bmRequestType` set to Device (`0x00` , the full `bmRequestType` being set to `0xC0`). While these calls are issued by Windows, and you shouldn't have to duplicate them, if you want an example of `Compatible ID` as well as `Extended Properties` Feature Descriptor retrieval using libusb, see the `read_ms_winsub_feature_descriptors()` function in the `xusb.c` sample source.

Thus, the second part of making your device WCID is ensuring that your firmware answers a Vendor Request with `wIndex=0x0004` and `bRequest=0x##` , to return a Compatible ID.

In its simplest form, and as documented here the format of the data packet is as follows:

Table 2: Microsoft Compatible ID Feature Descriptor

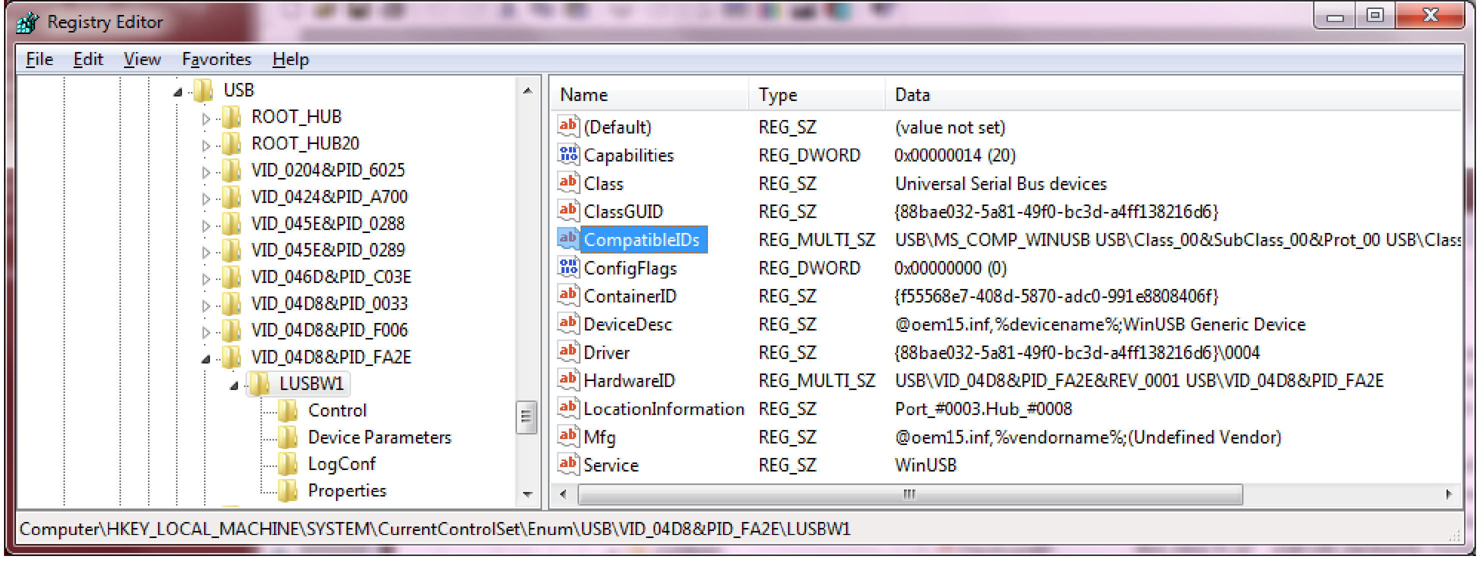
Value	Type	Description
`0x28, 0x00, 0x00, 0x00`	DWORD (LE)	Descriptor length (40 bytes)
`0x00, 0x01`	BCD WORD (LE)	Version (`'1.0'`)
`0x04, 0x00`	WORD (LE)	Compatibility ID

		Descriptor index (`0x0004`)
`0x01`	**BYTE**	Number of sections (1)
`0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00`	7 BYTES	Reserved
`0x00`	BYTE	Interface Number (Interface #0)
`0x01`	BYTE	Reserved
`0x57, 0x49, 0x4E, 0x55, 0x53, 0x42, 0x00, 0x00`	8 BYTES (NUL-terminated?) ASCII String	Compatible ID (`"WINUSB\0\0"`)
`0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00`	8 BYTES (NUL-terminated?) ASCII String	Sub-Compatible ID (unused)
`0x00, 0x00, 0x00, 0x00, 0x00, 0x00`	6 BYTES	Reserved

You can then check if the Compatible ID was properly read from your device by browsing the registry's `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB` tree at the location of your device. Similarly to the `osvc` key which is created on first insertion, you should find an entry with the VID and PID of your device under `USB` (eg: `VID_04D8&PID_FA2E`).

Under this entry, possibly under an additional sub-entry, you should verify the content of the `CompatibleIDs` key (which is a `REG_MULTI_SZ`). If Windows was able to read the `Compatible ID` from your device, this key should include a `USB\MS_COMP_XXXXXXXX` string, where `XXXXXXXX` is what was defined in your report. For instance, if you defined `LIBUSB0` , you should get `USB\MS_COMP_LIBUSB0` there.

Here is an example of the registry entry created from the Compatible ID Feature Descriptor (`"WINUSB"`):



Name	Type	Data
(Default)	REG_SZ	(value not set)
Capabilities	REG_DWORD	0x00000014 (20)
Class	REG_SZ	Universal Serial Bus devices
ClassGUID	REG_SZ	{88bae032-5a81-49f0-bc3d-a4ff138216d6}
CompatibleIDs	REG_MULTI_SZ	USB\MS_COMP_WINUSB USB\Class_00&SubClass_00&Prot_00 USB\Class
ConfigFlags	REG_DWORD	0x00000000 (0)
ContainerID	REG_SZ	{f55568e7-408d-5870-adc0-991e8808406f}
DeviceDesc	REG_SZ	@oem15.inf,%devicename%;WinUSB Generic Device
Driver	REG_SZ	{88bae032-5a81-49f0-bc3d-a4ff138216d6}\0004
HardwareID	REG_MULTI_SZ	USB\VID_04D8&PID_FA2E&REV_0001 USB\VID_04D8&PID_FA2E
LocationInformation	REG_SZ	Port_#0003.Hub_#0008
Mfg	REG_SZ	@oem15.inf,%vendorname%;(Undefined Vendor)
Service	REG_SZ	WinUSB

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\VID_04D8&PID_FA2E\LUSBW1

If you can find an `USB\MS_COMP_XXXXXXXX` entry for your device (here `USB\MS_COMP_WINUSB`), then your device qualifies as a WCID device. In other words, defining a Microsoft OS String Descriptor and a Microsoft Compatible ID Feature Descriptor in your firmware is all that's needed to turn it into a WCID device and benefit from the automated driver installation feature of Windows.

Other Compatible IDs than `"WINSUB"` can also be used according to the driver you want to automate the installation of. The following table provides a reference of which Compatible ID string you should use, according to the type of application you want to use your device with:

Table 3: Compatible ID to Application Type

Compatible ID String	Application Type
`"WINUSB\0\0"`	libusb application LibUsbDotNet application native WinUSB application
`"LIBUSB0\0"`	libusb-win32 application LibUsbDotNet application
`"LIBUSBK\0"`	libusbK application LibUsbDotNet application

Windows Automated Driver Installation

The WCID automated driver installation process is best illustrated through the native automated WinUSB driver installation of Windows 8.

Here again, there are two requirements that must be met to allow for the automated installation of a device driver.

First, the inf file must include `USB\MS_COMP_XXXXXXXX` as a device identifier string. In Windows 8 (x64), if you look at the `winusb.inf` from your `\Windows\System32\DriverStore\FileRepository\winusb.inf_amd64_50cb26e12c0e99f2\` (or similar), you will find that this is indeed the case see:

```
[Generic.Section.NTamd64]
%USB\MS_COMP_WINUSB.DeviceDesc%=WINUSB,USB\MS_COMP_WINUSB

[Strings]
Generic.Mfg="WinUsb Device"
USB\MS_COMP_WINUSB.DeviceDesc="WinUsb Device"
```

In comparison, the `winusb.inf` you'll find at the same location in Windows 7 is missing a `[Manufacturer]` section pointing to a compatible ID (but the `winusbcompat.inf` downloaded from Windows Update has it).

Secondly, the driver must have been pre-installed onto the system, which mostly means that an entry for it must exist in `\Windows\System32\DriverStore\FileRepository\`. On non Windows 8 platforms, this pre-installation can be done through Windows Update when trying to locate a driver.

As both these conditions are met by default for the WinUSB driver on Windows 8, it becomes possible to plug in a WinUSB WCID device (i.e. a WCID device that returns `"WINUSB"` for the Compatible ID Descriptor) and have its driver installed automatically: once Windows has populated the `CompatibleIDs` list in the registry, it will search the inf repository for a match, and if it finds one, install the relevant driver.

This whole process is actually nothing new, as a the same occurs for USB Class devices, such as HID or Mass Storage, where the `USB\Class_...` entry is used for the match. For instance, if you look at `input.inf` from `DriverStore\` you would see:

```
[Standard.NTamd64]
; Generic HID Interface (HidUsb as service)
%HID.DeviceDesc% = HID_Inst,,GENERIC_HID_DEVICE,USB\Class_03&SubClass_01,USB\Class_03

[Strings]
HID.DeviceDesc  = "USB Input Device"
StdMfg          = "(Standard system devices)"
```

and `USB\Class_03` is what you would find as one of the `CompatibleIDs` string for an HID Joystick for instance.

Defining a Device Interface GUID or other device specific properties

If you read the Microsoft documentation, you will find that Microsoft OS Descriptors can be used for more than just WCID, at WCID is only a subset of it. For instance, they can be used to provide icons, URLs or other data.

One item that may be of interest to you, with regards to WCID devices, is the provision of additional device registry settings and especially the provision of the Device Interface GUID that you want your device to be accessed with. Just like with the Compatible ID, this too is done through a Feature Descriptor, called `Extended Properties Feature Descriptor` and located at `wIndex = 0x0005`. However, as opposed to the Compatible ID, this should be queried using an Interface Request with the full `bmRequestType` being set to `0xC1`.

IMPORTANT NOTE 1: There is a bug/limitation in WinUSB that will force the `wIndex` of any Interface Request to the interface number. This means that, if you are using WinUSB to verify the content of your Extended Properties Feature Descriptor, you won't be able to retrieve it (unless it is only defined for interface #5).
If you use WinUSB on Windows to validate your descriptors, our advice then is to have your device firmware answer both Device and Interface requests when the Extended Properties Feature Descriptor is queried, so that you can use a Device Request rather than an Interface Request to validate that your descriptor is set properly. For reference, the latest `xusb` sample application from libusb has a `-w` switch that will force the `wIndex = 0x0005` to use the Device mode.

IMPORTANT NOTE 2: The official Microsoft documentation shows the definition of a `DeviceInterfaceGUID` as a `REG_SZ`, and this is what the sample firmwares we provide at the end are defined with. However, if you are using a device with multiple interfaces, you may run into an issue where issuing a registry query to get the Device Interface GUIDs will not return the GUID of the WCID interface when it isn't the first one, **unless** you set your Extended Properties to return a `DeviceInterfaceGUIDs` property (with an **s**) as a `REG_MULTI_SZ`, to define your GUID. To make matters worse, we have gotten reports that trying to use `DeviceInterfaceGUIDs` with a single interface device (to define a single GUID), **may not work**, and that you therefore should:

- Use `DeviceInterfaceGUID` with a single NUL terminated GUID if your device has only one interface

Table 4a: Microsoft Extended Properties Feature Descriptor

Value	TYPE	Description
`0x8E, 0x00, 0x00, 0x00`	DWORD (LE)	Descriptor length (142 bytes)
`0x00, 0x01`	BCD WORD (LE)	Version (`'1.0'`)
`0x05, 0x00`	WORD (LE)	Extended Property Descriptor index (`5`)
`0x01, 0x00`	**WORD** (LE)	Number of sections (`1`)
`0x84, 0x00, 0x00, 0x00`	DWORD (LE)	Size of the property section (132 bytes)
`0x01, 0x00, 0x00, 0x00`	DWORD (LE)	Property data type (`1` = Unicode

		REG_SZ, see table below)
`0x28, 0x00`	WORD (LE)	Property name length (40 bytes)

`0x44, 0x00, 0x65, 0x00,`
`(...)`
`0x74, 0x00, 0x00, 0x00`
	NUL-terminated
	Unicode String (LE)
		Property name
		`"DeviceInterfaceGUID"`

`0x4e, 0x00, 0x00, 0x00`	DWORD (LE)	Property data length (78 bytes)

`0x7b, 0x00, 0x46, 0x00,`
`(...)`
`0x7d, 0x00, 0x00, 0x00`
	NUL-terminated
	Unicode String (LE)
		Property name
		`"{xxxxxxxx-xxxx-`
		`xxxx-xxxx-`
		`xxxxxxxxxxxx}\0"`

- Use `DeviceInterfaceGUIDs` with a list of at least two GUIDs, terminated with a double NUL if your device has two interfaces or more.

Table 4b: Microsoft Extended Properties Feature Descriptor

Value	TYPE	Description
`0xe0, 0x00, 0x00, 0x00`	DWORD (LE)	Descriptor length (224 bytes for 2 GUIDs)
`0x00, 0x01`	BCD WORD (LE)	Version (`'1.0'`)
`0x05, 0x00`	WORD (LE)	Extended Property Descriptor index (`5`)
`0x01, 0x00`	**WORD** (LE)	Number of sections (`1`)
`0xd6, 0x00, 0x00, 0x00`	DWORD (LE)	Size of the property section (214 bytes)
`0x07, 0x00, 0x00, 0x00`	DWORD (LE)	Property data type (`7` = Unicode **REG_MULTI_SZ**, see table below)
`0x2a, 0x00`	WORD (LE)	Property name length (42 bytes)

`0x44, 0x00, 0x65, 0x00,`
`(...)`
`0x73, 0x00, 0x00, 0x00`
	NUL-terminated
	Unicode String (LE)
		Property name
		`"DeviceInterfaceGUIDs"`

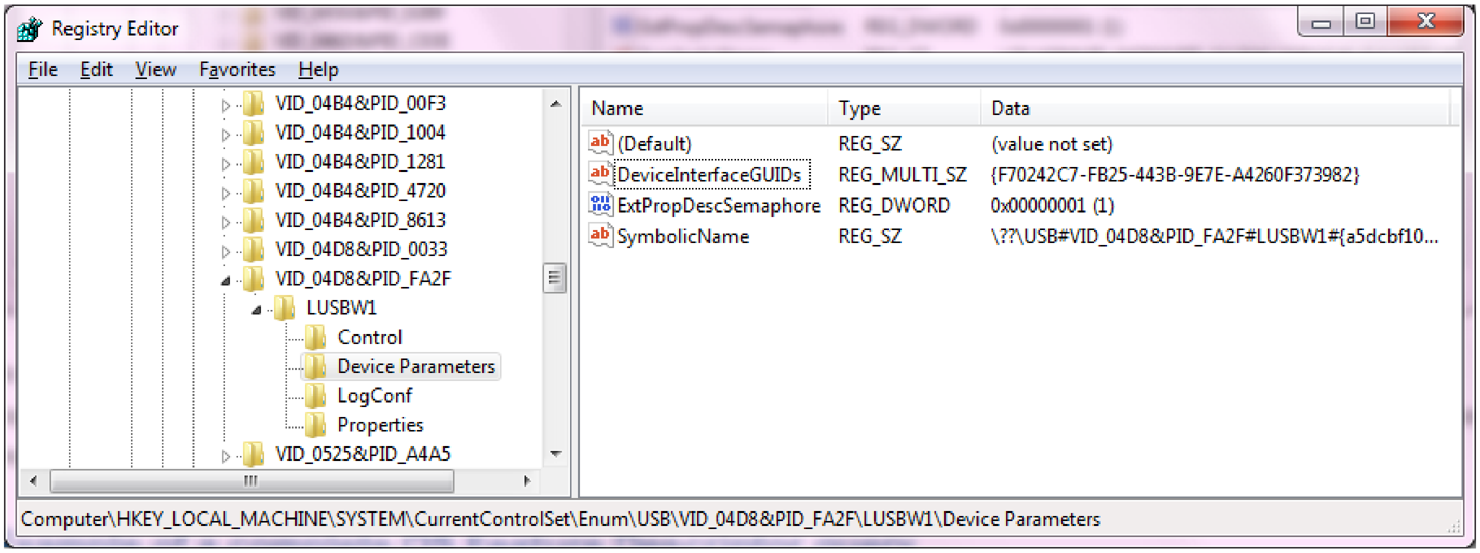
`0x9e, 0x00, 0x00, 0x00`	DWORD (LE)	Property data length (158 bytes)	
`0x7b, 0x00, 0x46, 0x00, (...) 0x00, 0x00, 0x00, 0x00`	**double NUL**-terminated Unicode String (LE)	Property name `"{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}\0{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}\0\0"`	

The property data types that you can use are defined in the following table:

Table 5: Data Types for the Microsoft Extended Properties Feature Descriptor

Value	Type
1	NUL-terminated Unicode String (REG_SZ)
2	NUL-terminated Unicode String, that may include environment variables (REG_EXPAND_SZ)
3	Binary Data (REG_BINARY)
4	DWORD Value, little endian (REG_DWORD_LITTLE_ENDIAN)
5	DWORD Value, big endian (REG_DWORD_BIG_ENDIAN)
6	NUL-terminated Unicode String, that may include a symbolic link (REG_LINK)
7	Multiple NUL-terminated Unicode Strings (REG_MULTI_SZ)

Should you define DeviceInterfaceGUIDs as highlighted above, or any other Extended Properties at index `0x0005` , you will find that it is copied into the registry, under the `Device Parameters` subsection of your device:



Name	Type	Data
(Default)	REG_SZ	(value not set)
DeviceInterfaceGUIDs	REG_MULTI_SZ	{F70242C7-FB25-443B-9E7E-A4260F373982}
ExtPropDescSemaphore	REG_DWORD	0x00000001 (1)
SymbolicName	REG_SZ	\??\USB#VID_04D8&PID_FA2F#LUSBW1#{a5dcbf10...

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\VID_04D8&PID_FA2F\LUSBW1\Device Parameters

As you can see, the `{F70242C7-FB25-443B-9E7E-A4260F373982}` that was defined in the Extended Properties Descriptors has properly been populated in the registry. You can therefore use the Extended Properties Descriptors to define any additional registry entry that your device may require, and then use the usual Microsoft APIs, such as SetupDi to read this data in your application.

Example of a complete OS Feature Descriptor query

For further details of the data that a WCID device, that also has the (optional) `DeviceInterfaceGUID` defined, should return, what follows is an example of what `xusb` against WCID-enabled Benchmark device. All the screenshots and data sections described above have been produced with this device. You can obtain a standalone `xusb` executable for Windows by downloading the latest Windows binary archive from http://libusb.info.

```
D:\libusb\Win32\Debug\examples>xusb -i 04d8:fa2f
Using libusb v1.0.21.11121

Opening device 04D8:FA2F...

Device properties:
        bus number: 2
         port path: 3 (from root hub)
             speed: 12 Mbit/s (USB FullSpeed)

Reading device descriptor:
            length: 18
      device class: 0
               S/N: 3
           VID:PID: 04D8:FA2F
         bcdDevice: 0001
   iMan:iProd:iSer: 1:2:3
          nb confs: 1

Reading BOS descriptor: no descriptor

Reading first configuration descriptor:
             nb interfaces: 1
              interface[0]: id = 5
interface[0].altsetting[0]: num endpoints = 2
   Class.SubClass.Protocol: 00.00.00
       endpoint[0].address: 01
           max packet size: 0020
          polling interval: 00
       endpoint[1].address: 81
           max packet size: 0020
          polling interval: 00

Claiming interface 0...

Reading string descriptors:
   String (0x01): "Travis Robinson"
   String (0x02): "Benchmark Device"
   String (0x03): "LUSBW1"
```

```
    String (0xEE): "MSFT100 "

Reading Extended Compat ID OS Feature Descriptor (wIndex = 0x0004):

  00000000  28 00 00 00 00 01 04 00 01 00 00 00 00 00 00 00  (...............
  00000010  00 01 57 49 4e 55 53 42 00 00 00 00 00 00 00 00  ..WINUSB........
  00000020  00 00 00 00 00 00 00 00                          ........

Reading Extended Properties OS Feature Descriptor (wIndex = 0x0005):

  00000000  8e 00 00 00 00 01 05 00 01 00 84 00 00 00 01 00  ................
  00000010  00 00 28 00 44 00 65 00 76 00 69 00 63 00 65 00  ..(.D.e.v.i.c.e.
  00000020  49 00 6e 00 74 00 65 00 72 00 66 00 61 00 63 00  I.n.t.e.r.f.a.c.
  00000030  65 00 47 00 55 00 49 00 44 00 00 00 4e 00 00 00  e.G.U.I.D...N...
  00000040  7b 00 46 00 37 00 30 00 32 00 34 00 32 00 43 00  {.F.7.0.2.4.2.C.
  00000050  37 00 2d 00 46 00 42 00 32 00 35 00 2d 00 34 00  7.-.F.B.2.5.-.4.
  00000060  34 00 33 00 42 00 2d 00 39 00 45 00 37 00 45 00  4.3.B.-.9.E.7.E.
  00000070  2d 00 41 00 34 00 32 00 36 00 30 00 46 00 33 00  -.A.4.2.6.0.F.3.
  00000080  37 00 33 00 39 00 38 00 32 00 7d 00 00 00        7.3.9.8.2.}...

Releasing interface 0...
Closing device...
```

WCID Advantages

- Actual Plug-and-Play for end-users
- **Zero** (Windows 8 native, Windows 7 through Windows Update) or **one** manual driver installation required, ever
- No need to go through WHQL and have a driver in Windows Update
- Reduced time to market and applications that can support future devices
- Much faster driver installation times (on Windows 8 or later, or once the WCID driver has been pre installed on Windows 7 or earlier)
- Works on **all** currently supported versions of Windows, starting with Windows XP SP2

WCID Drawbacks

- On Windows 7 or earlier, there is no known way to populate the device name during the driver installation, which means that WCID devices will be listed under the same generic name (`WinUsb Device`) in Device Manager. On Windows 8 or later, and if a Product String Descriptor has been defined, the device will be listed under this name.
- Only WinUSB on Windows Vista or later requires no user intervention whatsoever. For libusb-win32 or libusbK, or for WinUSB on Windows XP, the WCID driver must have been manually installed at least once (but only once)

WCID Device Examples:

PIC

For reference, an implementation of a typical Microchip PIC USB device firmware (with this diff providing an accurate indication of how much work is actually needed).

AVR (LUFA)

Also for reference, a simple LUFA based WCID firmware for Atmel AVR chips, that can be compiled using AVR Studio 5, is also provided here.

STM32

Another sample, for STM32 based (32-bit ARM Cortex) WCID devices, has been provided by **Darren Kattan** of Immense Networks. The sample is based on the USB Device Library V3.3.0, freely available on the ST website. Here is what Darren had to say about the samples:

> I literally made zero changes to core USB code. The stack is designed in such a way that there are callbacks to your user code for USB requests that the core code doesn't recognize. Nothing but 4 files had to be changed to make it work:
>
> - `USB_prop.c/.h`
> - `USB_desc.c/.h`
>
> I had to change the PLL settings to work with my hardware, so people working with dev boards will have to change them back. My target has a 16MHz clock instead of a 25MHz like most Connectivity Line dev boards have.

The v3.3.0 library archive can be found here and is labelled `STM32F1XX V3.3.0 (WinUSB).rar`. Also in the same directory are v2.1.0 library files labelled `STM32_USB-Host-Device_Lib_V2.1.0 (WINUSB HID).rar`.

Note that the only project that was modified in each library was the HID project.

Other

Finally, a Microsoft NetMon trace (using the USB extensions — for more details, please see this blog post) is also available here, that illustrates the USB queries your device will receive, and how it should answer them, the first time it is plugged on a Windows machine.

Aditional Links

- WinUSB Device (MSDN)
- Overview of Microsoft OS Descriptors by SourceQuest
- Microsoft-Defined USB Descriptors (MSDN)
- Microsoft USB FAQ (MSDN) → "What is a Microsoft OS Descriptor?"
- Microsoft OS Descriptors Overview (MSDN)

- How does USB stack enumerate a device? (MSDN)

Pages 9

Find a Page…

- **Home**
- **Compiling and debugging libwdi or Zadig**
- **FAQ**
- **Install**
- **Reuse**
- **Signed Driver Walkthrough**
- **Usage**
- **WCID Devices**
- **Zadig**

+ Add a custom sidebar

Clone this wiki locally

```
https://github.com/pbatard/libwdi.wiki.git
```