Loading a UASP Storage Driver as a Class Driver on Its Corresponding xHCI Stack

November 18, 2010

Abstract

This paper provides information about how to load a USB Attached SCSI Protocol (UASP) storage driver as a class driver on its corresponding Extensible Host Controller Interface (xHCI) stack for Windows® operating systems. It provides guidelines for hardware manufacturers and software vendors to develop their own USB driver stacks for xHCI controllers. By following these guidelines, developers of xHCI USB driver stacks can create their own UASP storage drivers that will automatically load when USB storage devices that support UASP are enumerated by their xHCI USB driver stacks.

This information applies to the following operating systems:
 Windows Server® 2008 R2
 Windows 7

 Windows Server 2008
 Windows Vista®
 Windows Server 2003

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the web at:
 <http://www.microsoft.com/whdc/connect/usb/loading-uasp-driver.mspx>

***Disclaimer****: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.*

*Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.*

*This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.*

© 2010 Microsoft Corporation. All rights reserved.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| November 18, 2010 | First publication |

Contents

[Background 3](#_Toc276468725)

[Solution 3](#_Toc276468726)

[Implementation 3](#_Toc276468727)

[The Compatible ID 4](#_Toc276468728)

[The Device ID 4](#_Toc276468729)

[The Hardware IDs 5](#_Toc276468730)

[Example 5](#_Toc276468731)

[Conclusion 5](#_Toc276468732)

[Resources 6](#_Toc276468733)

# Background

The Windows® operating system does not currently include native support for Extensible Host Controller Interface (xHCI) controllers (USB controllers that support SuperSpeed USB) or USB storage devices that conform to the USB Attached SCSI Protocol (UASP) specification. However, storage device manufacturers want to ensure that their new USB storage devices that support UASP will work on all systems, both those that support Enhanced Host Controller Interface (EHCI) controllers and those that support xHCI controllers. For systems that support EHCI controllers, the USB storage devices use the Microsoft® USB storage class driver that is included with Windows. But for systems that support xHCI controllers, no single UASP storage class driver currently exists.

Because Windows does not currently include native support for xHCI controllers, xHCI controller manufacturers are having USB driver stack vendors develop xHCI USB driver stacks for their xHCI controllers. A USB stack vendor could actually be a group within the same company as an xHCI controller manufacturer, or it could be an ISV that is hired by the xHCI controller manufacturer. Because each xHCI controller manufacturer is providing their own USB driver stack, each xHCI USB driver stack has its own proprietary streams interface. Without a standardized streams interface, it is difficult for storage device manufacturers to provide UASP storage drivers for their USB storage devices that will work on all xHCI controllers.

Storage device manufacturers and USB stack vendors expect Microsoft to release a standardized xHCI USB driver stack streams interface at some point in the future. This will enable interoperability between different UASP storage drivers and xHCI USB driver stacks. However, until such a standardized streams interface is released, another solution is required.

# Solution

A USB stack vendor can create an end-to-end solution by developing a UASP storage driver that works with their proprietary streams interface, where the UASP storage driver is automatically loaded on their xHCI USB driver stack. This UASP storage driver can be bundled with the xHCI USB driver stack software that ships with the corresponding xHCI controller.

When compatible UASP storage drivers are bundled with xHCI USB driver stack software, end users win. Regardless of the storage device or the xHCI controller manufacturer, the end user simply plugs in the storage device and the required drivers load automatically. The user does not need to download drivers from Windows Update or a support website, or install them from a CD. By bundling everything together, USB stack vendors can offer the end user a simple, hassle-free user experience.

# Implementation

When a USB driver stack reports specific Plug and Play (PnP) IDs for the USB devices that are attached to its respective USB host controller, it must include a unique short Bus ID string in the PnP ID. This string is used to identify the enumerator that is responsible for creating that PnP ID. In the examples below, USB\_VENDOR\_BUSID is used to represent this short string. USB stack vendors must use their own unique string in place of this example string for their USB driver stacks.

When an xHCI USB driver stack enumerates a USB storage device that supports UASP, it generates Device and Hardware IDs just like it would for any other device. Additionally, it should generate a Compatible ID that is specific to the UASP device class. It is this Compatible ID that makes it possible to load the UASP storage driver as a class driver. If the USB driver stack does not generate a correct Compatible ID for the USB storage device, then the stack-specific UASP storage driver will not automatically load.

## The Compatible ID

USB driver stacks must use the following format when they generate Compatible IDs for USB storage devices that support UASP.

USB\_VENDOR\_BUSID\Class\_08&SubClass\_06&Prot\_62

In this example, USB\_VENDOR\_BUSID represents the USB driver stack’s unique Bus ID string, and Class\_08&SubClass\_06&Prot\_62 is specific to the UASP device class.

The Compatible ID that is specified in the stack-specific UASP storage driver’s INF file must match this Compatible ID so that the UASP storage driver loads when a USB storage device that supports UASP is enumerated.

**Note:** The generic Bus ID string “USB” is reserved by Windows. Therefore, a USB driver stack must not generate the following Compatible ID for a USB device.

USB\Class\_08&SubClass\_06&Prot\_62

A USB driver stack should always use a unique Bus ID when generating Compatible IDs.

## The Device ID

USB driver stacks must use the following format when they generate Device IDs for the devices that they enumerate.

USB\_VENDOR\_BUSID\VID\_xxxx&PID\_xxxx

In this example, USB\_VENDOR\_BUSID represents the USB driver stack’s unique Bus ID string, and VID\_xxxx and PID\_xxxx represent the vendor ID and the product ID for the USB device.

This Device ID is used by Windows to generate a unique identifier that is associated with a specific device instance and a particular driver installation. Whenever a USB driver stack reports this unique identifier to Windows, the operating system will determine whether a device driver has already been installed for the device that matches that particular identifier, and if so, it will simply load that driver rather than search for a matching INF file. Therefore, it is important for this Device ID to be unique per xHCI USB driver stack. Otherwise, if a USB storage device that supports UASP is first enumerated by one xHCI USB driver stack, and the corresponding USB driver stack-specific UASP storage driver is installed and loaded for that device, and then that same device is enumerated by a separate xHCI USB driver stack on the same system, the USB driver stack-specific UASP storage driver for the first USB driver stack would still load for the device, which would not necessarily be compatible with the second USB driver stack.

## The Hardware IDs

USB driver stacks must use the following format when they generate Hardware IDs for the devices that they enumerate.

USB\_VENDOR\_BUSID\VID\_xxxx&PID\_xxxx&REV\_xxxx

USB\_VENDOR\_BUSID\VID\_xxxx&PID\_xxxx

USB\VID\_xxxx&PID\_xxxx&REV\_xxxx

USB\VID\_xxxx&PID\_xxxx

In this example, USB\_VENDOR\_BUSID represents the USB driver stack’s unique Bus ID string, and VID\_xxxx, PID\_xxxx, and REV\_xxxx represent the vendor ID, the product ID, and the revision ID for the USB device. Note that the USB driver stack must generate Hardware IDs using both the USB driver stack’s unique Bus ID string as well as the generic Bus ID string “USB”.

## Example

Fabrikam, an xHCI controller manufacturer, provides an xHCI USB driver stack and a UASP storage driver with their xHCI controllers. When the Fabrikam USB driver stack enumerates a USB storage device that supports UASP, it generates the proper Device and Hardware IDs for the device as well as the following Compatible ID.

USB\_FBRKM\Class\_08&SubClass\_06&Prot\_62

The Fabrikam UASP storage driver’s INF file matches on the Compatible ID that was reported by the USB driver stack instead of matching on a specific Device ID or Hardware ID. By matching on the Compatible ID, Fabrikam’s UASP storage driver will load for any USB storage device that supports UASP, not just a single storage product from a specific USB storage device manufacturer.

This enables Fabrikam to offer an end-to-end solution for storage device manufacturers and a great user experience for end users. The storage device manufacturers are not required to provide any drivers for their USB storage devices, and the end users are not required to download any drivers to use these USB storage devices on their systems.

# Conclusion

Microsoft believes in the future of both SuperSpeed USB (USB 3.0) and USB storage devices that support UASP. This paper provides a temporary solution for supporting UASP storage drivers on xHCI driver stacks that meets today’s requirements. To address future requirements, Microsoft is considering defining its own USB driver stack streams interface. However, Microsoft has made no formal commitment to provide such an interface at this time.

# Resources

Overview of INF Files

 <http://msdn.microsoft.com/en-us/library/ff549520.aspx>

INF File Sections and Directives

<http://msdn.microsoft.com/en-us/library/ff547433.aspx>

How does USB stack enumerate a device?

<http://blogs.msdn.com/b/usbcoreblog/archive/2009/10/31/how-does-usb-stack-enumerate-a-device.aspx>