Multifunction Device Support and Device Container Groupings in Windows 7

Guidelines for IHVs, OEMs, and Hardware Developers

October 29, 2008

Abstract

This white paper provides information about the multifunction device support and device container groupings for Microsoft Windows® 7 operating system. It provides guidelines that IHVs and OEMs should follow when designing and developing new hardware for the Windows 7 platform.

The improved support for multifunction devices and device container grouping is discussed in detail, including:

* An overview of the architecture.
* Algorithms and heuristics employed by the operating system for multifunction device detection.
* Recommendations to hardware and driver developers to ensure their devices work well with Windows 7.

This information applies for the following operating systems:
 Windows 7

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

For the latest information, see:
 <http://www.microsoft.com/Device/DeviceExperience/ContainerIDs.mspx>

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows NT, Windows Server, Rally, Windows 7 and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

[Introduction 5](#_Toc213464696)

[Terminology and Definitions 5](#_Toc213464697)

[Devices in Windows Vista versus Windows 7 6](#_Toc213464698)

[Grouping Functionality into a Device: the Container ID 7](#_Toc213464699)

[Overview 7](#_Toc213464700)

[How the ContainerID Is Assigned 7](#_Toc213464701)

[Generating ContainerID from the Bus-Specific Unique Identifier 8](#_Toc213464702)

[Generating ContainerID from the Removable Device Capability 11](#_Toc213464703)

[Including ContainerID Support in a Custom Bus Driver 13](#_Toc213464704)

[Setting the Removable Device Capability 14](#_Toc213464705)

[Handling BusQueryContainerID 14](#_Toc213464706)

[Overriding the Removable Capability in Legacy Devices 15](#_Toc213464707)

[DeviceOverrides Registry Key 18](#_Toc213464708)

[Comments 18](#_Toc213464709)

[“*HardwareID”* or “*CompatibleID*” Registry Key 18](#_Toc213464710)

[Comments 18](#_Toc213464711)

[LocationPaths Registry Key 19](#_Toc213464712)

[Comments 19](#_Toc213464713)

[ChildLocationPaths Registry Key 19](#_Toc213464714)

[Comments 20](#_Toc213464715)

[\* Registry Key 20](#_Toc213464716)

[Comments 20](#_Toc213464717)

[“*Location Path*” Registry Key 20](#_Toc213464718)

[Comments 21](#_Toc213464719)

[Removable Registry Value 22](#_Toc213464720)

[Comments 22](#_Toc213464721)

[Device Override Examples 22](#_Toc213464722)

[Example 1 24](#_Toc213464723)

[Example 2 25](#_Toc213464724)

[Example 3 26](#_Toc213464725)

[Example 4 27](#_Toc213464726)

[Verifying Correct Implementation of ContainerID 28](#_Toc213464727)

[Devices and Printers Folder 28](#_Toc213464728)

[Call to Action 30](#_Toc213464729)

[Resources 31](#_Toc213464730)

#

# Introduction

The world of hardware and devices is rapidly changing. Today’s devices are integrating a richer and ever increasing set of functionality into the hardware. Examples of this multifunction integration are everywhere:

* Multifunction printer, scanner, and copier products.
* Smart storage devices with integrated security.
* Cellular phones which include media playing capabilities.

One goal of the Windows 7 operating system is to provide improved support for multifunction devices. This improved support will stretch throughout the OS platform, providing:

* Enhancements to the Windows Plug and Play (PnP) infrastructure to detect and group the functions in a device.
* A new user interface for users to view and interact with their devices, where devices will appear as they do on the user’s physical desktop. This user interface will incorporate detailed icons and descriptions for the device, and expose device functionality to the user.

This paper will cover the changes to the Windows PnP subsystem to detect and group device functionality. Additional Windows 7 white papers and developer documentation will cover the other platform enhancements previously mentioned.

# Terminology and Definitions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Term | Definition |  |  |  |
| Functional device instance | A single device instance as seen by PnP. A functional device instance provides one of possibly many functional end points of a physical device. |
| Devnode | An internal structure that represents a single device instance on the system. A devnode contains the device stack and information about the device, such as the state and attributes of the device as seen by Windows PnP. |
| Multifunction device | A device that is perceived as a single “piece of plastic” to the user, but which may contain many functional device instances. These functions are physically contained within the device enclosure or chassis. |
| ContainerID | An identifier associated with a functional device instance that is shared among all functional device instances contained within a single multifunction device. |
| Plug and Play Extensions (PnP-X) | An extension of Windows Plug and Play, which supports network-connected devices. PnP-X currently supports a Plug and Play experience for devices which implement the Device Profile for Web Services (DPWS) and Universal Plug and Play (UPnP). PnP-X and DPWS are part of the Microsoft Windows Rally™ device connectivity service |

# Devices in Windows Vista versus Windows 7

A comparison of how devices are represented by PnP in Windows Vista® and Windows 7 will illustrate the new platform support for multifunction devices.

In the Windows family of operating system, devices are basically a collection of functional device instances, each of which represent a functional end point allowing some form of communication to the device. The term *devnode* is often used to refer to the driver stack for such a functional end point, in addition to the properties describing the end point and its associated state. For example, a multifunction printer, scanner, and fax device may have multiple PnP devnodes, one for each of the functional end points in the device. (That is, the printer, scanner, and fax are integrated into the single “piece of plastic” that the user physically sees as the device.)

In Windows Vista and prior PnP-enabled versions of Windows, each functional end point has a PnP devnode associated with it. Windows platform components and third-party applications can query PnP devnodes for device status and information, and communicate with device hardware though interfaces exposed from functional end points.

 In the case of a single function device, there is a single devnode that contains all of the information relating to that device. Likewise, a multifunction device has multiple devnodes associated with it. However, Windows lacks the capability to recognize that a group of devnodes originated from the same physical device. Each of the individual devnodes that belong to the same multifunction device does not include any identification information that would allow Windows PnP to group them together as a single device. Thus it is not possible to have a holistic view of the device and the functions provided by an individual physical device instance.

Windows 7 extends PnP to allow it to recognize that a group of devnodes originated from and belong to a particular physical device instance. This new extension to PnP is simply a layer on top of the existing PnP devnode model. By grouping all sets of devnodes that originated from single device instances, it is possible to understand the relationship among their different device functions. This makes it possible to present the user (or applications) with a device-centric view of devices instead of the traditional function-centric view. This provides a more natural representation of a device to the user, who is able to see all of the functionality exposed by the physical device grouped together under a single object.

# Grouping Functionality into a Device: the Container ID

With Windows 7, a new PnP property has been introduced to allow Windows components to group together all devnodes originating from the same physical device container, enclosure, or chassis. This new property is the *Container ID*.

## Overview

The ContainerID is a property of every devnode in PnP. Its type is a Globally Unique Identifier (GUID). The ContainerID is unique to a single instance of a physical device, where all device function devnodes that belong to that instance of the physical device share the same ContainerID value. For example:

Figure . ContainerIDs for devnodes on a multifunction device

 shows a multifunction device (MFD) connected to a Windows 7 PC. Upon initial connection to the PC, the bus driver detects the presence of the new MFD device. PnP is notified and begins the installation process. As part of the installation process, each functional device in the larger multifunction device is enumerated by the bus driver. For each of these functional devices, PnP creates a devnode and stores bus reported properties, such as hardware IDs, compatible IDs, and device capabilities. The ContainerID is one of these properties. Each of the devnodes originating from the same physical device will be assigned the same ContainerID value during enumeration.

## How the ContainerID Is Assigned

The ContainerID is assigned to a devnode using one of two mechanisms:

1. Retrieving a ContainerID from a bus driver. In this case, the bus driver acquired the ContainerID from the physical device hardware, or generated a ContainerID using a unique identifier from the device.
2. Examining the Removable device capability of all devnodes comprising the device.

When assigning a ContainerID to a devnode, Windows PnP first checks if the bus driver of the devnode can provide a ContainerID. In this case, a bus driver can either acquire a genuine ContainerID that was embedded in physical device hardware, or use a bus-specific unique identifier from the device hardware to generate a ContainerID. Some examples of a bus-specific unique identifier are a device serial number or a Media Access Control (MAC) address in the firmware of the device. The unique identifier varies depending upon the bus to which the device is attached. Details about the bus-specific unique identifiers that Windows recognizes are provided in the following sections.

If a bus driver cannot provide a ContainerID for a devnode that it is enumerating, Windows PnP will use the Removable device capability to determine which devnodes belong to a device container grouping. In this case, each devnode belonging to a device container will be assigned a newly generated ContainerID that is only unique to the host PC. In other words, the same device will have a different and non-deterministic ContainerID assigned to all of its devnodes when connected to a different PC.

### Generating ContainerID from the Bus-Specific Unique Identifier

The preferred way to generate a ContainerID for a device is based on a bus-specific unique identifier. This is the most precise and reliable mechanism for generating a ContainerID, and will always be used if:

1. The device contains a bus-specific unique identifier.
2. The bus driver recognizes this unique identifier as present and well formatted.
3. The bus driver can reliably hash the unique identifier into a GUID, and returns this GUID in response to the IRP\_MN\_QUERY\_ID function code with the BusQueryContainerID subtype.

Windows 7 includes inbox support for several of the most common bus types, including USB, Bluetooth, and PnP-X (IP connected) devices. For devices that utilize these transports, the device only needs to include the bus-specific unique identifier for that transport. The supplied Windows bus drivers will do the work of reading unique identifiers from devices and creating ContainerIDs.

#### Universal Serial Bus Devices

For devices connected to the PC using Universal Serial Bus (USB), there are three supported methods for generating the ContainerID for a device.

1. A USB device can report a ContainerID by providing a Microsoft® OS ContainerID Descriptor. The ContainerID Descriptor is a new Microsoft OS Feature Descriptor in Windows 7. The primary advantages to using a ContainerID Descriptor are that the IHV can precisely specify the ContainerID value for a device, and the ContainerID value for a device is unique and will not change between different PCs that the device could be connected to.

Complete information about Microsoft OS Descriptors and implementation of the new ContainerID Descriptor can be found on the [Microsoft Windows Hardware Developer Central](http://www.microsoft.com/whdc/connect/usb/os_desc.mspx) website. This topic is listed in “Resources” at the end of this paper.

1. If the USB device does not report a Microsoft OS ContainerID Descriptor but does report a serial number and reports itself as Removable, the USB hub driver will generate a ContainerID using a combination of the device serial number, vendor ID, product ID, and revision number. This ContainerID will be generated using the UUID Version 5 (SHA-1 hash) algorithm under a USB-specific namespace. The generated ContainerID will be unique, provided the IHV provides a unique serial number on each individual device.
2. If the device does not contain a Microsoft OS ContainerID Descriptor or report a serial number, PnP will check the Removable device capability for the devnodes enumerated by the device. This mechanism for generating the ContainerID exists to provide backward compatibility for devices that are unable to provide a Microsoft OS ContainerID Descriptor or a USB serial number.

However, it is important to recognize that the USB hub driver uses available removability information from the physical USB hardware in order to report a more accurate Removable device capability for devices connected to each of its internally or externally facing ports. Refer to the “[Generating ContainerID from the Removable Device Capability](#_Generating_ContainerID_from)” for more information.

For devices that are not integrated into the PC (that isall external devices), it is a best practice to always provide a Microsoft OS ContainerID Descriptor and a serial number in the USB device hardware. This will ensure that the Windows PnP infrastructure is able to correctly group all of the device functions exposed by the device. Features in the Windows 7 platform and future Windows releases will rely on the proper grouping of device functions. Following this practice will provide the best user experience for devices on the Windows platform.

If the device is to be integrated into the PC (for example, an internal Bluetooth radio or an integrated webcam), the device should rely on the Removable capability to ensure that the device is grouped correctly with the PC. In this case, the device must not report a Microsoft OS ContainerID Descriptor or serial number for the device.

#### Bluetooth Devices

For Bluetooth devices connected to the PC, the Media Access Control (MAC) address of the Bluetooth device is used as the bus-specific unique identifier of the device. The Bluetooth driver stack uses the MAC address as a seed value to generate a ContainerID for the device. This ContainerID value is supplied by the Bluetooth bus driver for each Bluetooth devnode enumerated for a physical device.

Bluetooth devices often implement Bluetooth-specific services. These services are not installed as Windows PnP devices, and therefore do not have associated devnodes. These services are, however, effectively functional device instances, as they provide specific functionality and enable communication with the Bluetooth device. Windows regards Bluetooth services as functional device interfaces, and new infrastructure exists to group the Bluetooth services along with the Bluetooth devnodes for a device.

All Bluetooth devices are required to include a MAC address. As a result, the ContainerID for Bluetooth devnodes and services will always be based on the MAC address value. As a result, the Removable device capability is never used as a basis for ContainerID generation for Bluetooth devices.

To ensure that a unique ContainerID is generated for every device, the Bluetooth device MAC address should be unique among all Bluetooth device instances.

#### Plug and Play Extensions (PnP-X) for Network Connected Devices

PnP-X extends Windows PnP to support devices connected to the PC via an IP network. PnP-X devices have the ability to specify a ContainerID as an XML element in their device metadata. Two protocols are supported: Device Profile on Web Services (DPWS) and Universal Plug and Play (UPnP). More information about PnP-X, DPWS and UPnP can be acquired through links listed in “Resources” at the end of this paper.

If a PnP-X device does not specify a ContainerID in the DPWS device metadata or the UPnP device description document, Windows PnP will generate a ContainerID for the device.

##### Specifying a ContainerID for DPWS Devices

Devices which implement the Device Profile for Web Services (DPWS) and support PnP-X can choose to specify a ContainerID. This is accomplished by including the <ContainerId> XML element in the device metadata document for the device.

| Namespace | <http://schemas.microsoft.com/windows/2008/09/devicefoundation> |
| --- | --- |
| **XML element prototype** | <df:ContainerId xmlns:df=”<http://schemas.microsoft.com/windows/2008/09/devicefoundation>”> *xs:string* </df:ContainerId> |

The <ContainerId> XML element type is a string, for which the value is a GUID of the form *{00000000-0000-0000-0000-000000000000}*. For example:

|  |
| --- |
| <df:ContainerId xmlns:df=”[http://schemas.microsoft.com/windows/2008/09/devicefoundation](https://mail.exchange.microsoft.com/OWA/redir.aspx?C=2f9e5565fff246f88c0e5e0af1888fc6&URL=http%3a%2f%2fschemas.microsoft.com%2fwindows%2f2008%2f09%2fdevicefoundation)”> {101392d0-5e91-11dd-ad8b-0800200c9a66}</df:ContainerId> |

##### Specifying a ContainerID for UPnP Devices

Devices that implement the UPnP protocol and support PnP-Xcan choose to specify a ContainerID. This is accomplished by including the <ContainerId> XML element in the device description document for the device.

| Namespace | <http://schemas.microsoft.com/windows/2008/09/devicefoundation> |
| --- | --- |
| **XML element prototype** | <df:ContainerId xmlns:df=”<http://schemas.microsoft.com/windows/2008/09/devicefoundation>”> *xs:string* </df:ContainerId> |

The ContainerId XML element type is a string, for which the value is a GUID of the form *{00000000-0000-0000-0000-000000000000}*. For example:

|  |
| --- |
| <df:ContainerId xmlns:df=”[http://schemas.microsoft.com/windows/2008/09/devicefoundation](https://mail.exchange.microsoft.com/OWA/redir.aspx?C=2f9e5565fff246f88c0e5e0af1888fc6&URL=http%3a%2f%2fschemas.microsoft.com%2fwindows%2f2008%2f09%2fdevicefoundation)”> {101392d0-5e91-11dd-ad8b-0800200c9a66}</df:ContainerId> |

### Generating ContainerID from the Removable Device Capability

When a device does not embed a ContainerID or unique identifier in its hardware, PnP will examine the Removable device capability on each devnode to determine how the devnodes should be grouped into a device. Before examining this algorithm, it is necessary to understand the Removable device capability and when a devnode should report itself as removable.

The Removable device capability is a bit that can be set in the DEVICE\_CAPABILITIES structure that is populated by drivers in response to the IRP\_MN\_QUERY\_CAPABILITIES function code for a given devnode. (See the MSDN Library for details on the [DEVICE\_CAPABILITIES](http://msdn.microsoft.com/en-us/library/aa491648.aspx) structure.)

The Removable device capability is set on a devnode when the devnode and all of its child devnodes make up a device which can be physically removed, disconnected, or unplugged from its parent devnode while the PC is running. Typically, a devnode should be marked as removable if it is the topmost devnode in a devnode topology.

Setting the removable device capability correctly on a devnode is important. If the device does not include a unique identifier (as described previously), Windows PnP uses this capability to determine which devnodes belong to a device.

For example, suppose that a single function device, such as a mouse, is connected to the PC via USB. In this case, the USB bus will see that a new device has arrived, detect that it is a USB human interface device (HID), and create a USB HID devnode for the device. The HID devnode will further detect that the HID device is a mouse, and will create a child devnode for a HID-compliant mouse. At this point, the mouse is installed, and is functional on the system. Both of the new devnodes use independent driver stacks. Recall that the topmost devnode of the device should be set as removable, while its child devnodes should not be set as removable. Therefore, the USB HID devnode should have Removable set to TRUE, while its child devnode (the HID-compliant mouse) should have Removable set to FALSE.

The following Device Manager screen shot shows the devnode topology for a generic USB mouse, indicating which devnodes of the mouse are marked as removable:



**Removable** = TRUE

**Removable** = FALSE

Figure . Devnode topology for a USB mouse as seen in Device Manager

In the absence of a unique identifier for the device (as in the case of the mouse example), PnP will use the Removable capability to determine which devnodes belong to a device. This is done by applying the following heuristic to each devnode as it is created:

1. If the bus driver of the device reported a genuine ContainerID (for example, Microsoft OS Descriptor) or a generated ContainerID based on a unique identifier in the hardware (for example, serial number, MAC address), then assign this ContainerID to the devnode.
2. If the devnode has the Removable capability set to TRUE, generate a new ContainerID for the devnode.
3. If the devnode has the Removable capability set to FALSE, inherit the ContainerID of its parent devnode.

A devnode cannot enumerate child devnodes until it is initialized and its driver stack is started. When ContainerID assignment is part of the devnode initialization, the devnode is ready to propagate its ContainerID down to any of its non-removable children as they are enumerated.

A devnode with the removable device capability is considered to be the topmost devnode for the device, and a ContainerID is generated and applied to this devnode. All of the children of this topmost devnode will inherit the same ContainerID, unless they are physically detachable from their parent devnode.

Applying this heuristic to the USB mouse example results in the following actions:

1. The USB Human Interface Device devnode is created. No unique identifier exists in the hardware of this device.
	1. The Removable capability is set to TRUE on this devnode, because its parent USB hub devnode recognized that it was plugged into an externally facing USB port.
	2. Create a ContainerID for this devnode, as it is the topmost devnode of a removable device.
2. The HID-compliant mousedevnode is created. No unique identifier exists in the hardware of this device.
	1. The Removable capability is set to FALSE on this devnode because its parent USB HID devnode reports all of its children as non-removable. HID-compliant mouse has no notion of removability.
	2. Inherit the ContainerID of the parent devnode.

After applying the heuristic, the same ContainerID has been assigned to each devnode belonging to the mouse. PnP was able to successfully group the devnodes into a logical device, even in the absence of a unique identifier for the device.

It is important to note that the success of this heuristic relies on a given bus driver to correctly report the removable device capability for each devnode that it reports. For this reason, it is important that the removable capability be reported accurately.

## Including ContainerID Support in a Custom Bus Driver

In Windows 7, all Microsoft-supplied bus drivers are device-container aware. If a device-specific unique identifier is available, these bus drivers will use that identifier to create a ContainerID for each devnode in a device. If a ContainerID cannot be generated based on a unique identifier, the reported Removable device capability is used to guide PnP as it creates and/or propagates ContainerIDs for devnodes.

Third party bus drivers should be written to be device-container aware. Supporting device containers in a custom bus driver requires that the bus driver do the following:

1. Correctly set the Removable device capability on each devnode enumerated by the bus driver, such that Removable is only set to TRUE for devnodes that represent physically detachable device connection end points.
2. If the bus supports a ContainerID or unique identifier in the device hardware, the bus driver can handle the BusQueryContainerID subtype for the IRP\_MN\_QUERY\_ID function code.

### Setting the Removable Device Capability

The guidelines for setting the Removable device capability for a devnode enumerated by a custom bus driver are the same as for other bus drivers. Refer to a previous section titled “” for more information about which devnodes should be marked Removable for a device.

### Handling BusQueryContainerID

The following code sample demonstrates how a bus driver should return a ContainerID for a devnode. Recall that a ContainerID should only be reported for devices which are external to the PC. Devices which are internal to the PC should be reported as not Removable, which will enable PnP to group the device with the PC. In this example, the **PdoHandleIrpQueryId** function handles the IRP\_MN\_QUERY\_ID function code for a WDM bus driver PDO.

|  |
| --- |
| NTSTATUSPdoHandleIrpQueryId( PDEVICE\_OBJECT Pdo,PIRP           Irp){NTSTATUS           status;PIO\_STACK\_LOCATION ioStack;GUID               containerId;UNICODE\_STRING     unicodeContainerId;ioStack = IoGetCurrentIrpStackLocation(Irp);switch (ioStack->Parameters.QueryId.IdType) { case BusQueryHardwareIDs:...break; case BusQueryInstanceID:...break; ... case BusQueryContainerID:status = GET\_CONTAINERID\_FROM\_PDO(Pdo, &containerId); if (NT\_SUCCESS(status)) {status = RtlStringFromGUID(&containerId, &unicodeContainerId); if (NT\_SUCCESS(nts)) {Irp->IoStatus.Information = (ULONG\_PTR)unicodeContainerId.Buffer;status = STATUS\_SUCCESS;}}break; default:...break;} Irp->IoStatus.Status = status;IoCompleteRequest(Irp, IO\_NO\_INCREMENT); return status;} |
|  |

When the BusQueryContainerID query ID subtype is received, the bus driver must return Unicode string buffer allocated from a page pool. This buffer contains a string representation of the ContainerID GUID that should be assigned to the devnode.

The example shows an undefined call into GET\_CONTAINERID\_FROM\_PDO to get a ContainerID GUID. The details of where this ContainerID comes from and how it is generated are bus-specific. As with other ID types that are queried often, such as BusQueryHardwareIDs and BusQueryInstanceID, it is good practice to have the ContainerID ready and stored against a bus-specific PDO extension to allow for efficient handling of IRPs.

# Overriding the Removable Capability in Legacy Devices

The algorithm to group devnodes into a logical device, based on the Removable device capability, partially exists to support legacy devices which do not provide recognized device-specific unique identifiers in their hardware. New devices should provide a device-specific unique identifier and set the Removable device capability correctly for the device.

However, there are legacy devices that may not report the Removable device capability correctly. As a result, these devices will not be grouped into their expected logical device containers. A mechanism to override the bus reported Removable device capability for a devnode is provided to address this case.

This device override mechanism is a registry-based lookup table that consists of registry keys that map to specific devices. This override table is maintained under the following registry key:

**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceOverrides**

Subkey entries under the **DeviceOverrides** key are device hardware IDs and/or compatible IDs, where ‘\’ path separator characters are replaced with ‘#’ pound characters. These altered hardware and compatible IDs correspond to devices on which the Removable device capability will be overridden for ContainerID generation purposes.

Subkey entries in the override table do not actually change the global state of the Removable device capability on the devnode, but rather only affect its evaluation for ContainerID generation. Additional subkeys under the hardware and/or compatible ID subkeys allow for finer grain specification of exactly which devnodes to override based on their connected locations within the PC.

The following tree shows the structure of the **DeviceOverrides** registry key and subkeys. Usage of each subkey is explained in detail.



Figure . Removable override registry structure

## DeviceOverrides Registry Key

| Registry key name | DeviceOverrides |
| --- | --- |
| **Required/optional** | Required |
| **Format requirements** | None |
| **Valid subkeys** | * **HardwareID**
* **CompatibleID**

(See the following subkey format requirements and examples) |

### Comments

The **DeviceOverrides** registry key indicates that one or more overrides for the Removable device capability may exist on this PC. Device overrides allow an IHV/OEM to change the interpreted value of the Removable device capability on a devnode or group of devnodes. This is useful for legacy devices or third- party hardware components that may not report the Removable device capability correctly, and in turn cause the ContainerID generation heuristic to incorrectly group the related devnodes. The existence of a Removable device capability registry override does not change the global state of the Removable device capability on a devnode.

The **DeviceOverrides** registry key must be created by the first device override that is added to the table. It may not exist by default on a clean OS installation.

## “*HardwareID”* or “*CompatibleID*” Registry Key

| Registry key name | Valid “*HardwareID*” or “*CompatibleID*” values. |
| --- | --- |
| **Required/optional** | Required |
| **Format requirements** | * Must include the bus prefix of the **HardwareID** or **CompatibleID**
* All ‘\’ characters must be replaced with ‘#’
 |
| **Examples** | * USB#VID\_1234&PID\_ABCD&REV\_0001
* PCI#VEN\_ABCD&DEV\_1234&SUBSYS\_000
 |
| **Valid subkeys** | * **LocationPaths**
* **ChildLocationPaths**
 |

### Comments

The **HardwareID** or **CompatibleID** registry subkey specifies a device for which a Removable device override will be applied.

Notice in the examples that the bus prefix for the device is present, and that the backslash character “\” has been replaced with the pound sign “#”. Because the backslash character is not a valid character in a registry subkey name, you must replace it with the pound sign when specifying a bus prefix.

In general, the most specific device hardware ID should be used to identify a device, rather than a less specific hardware or compatible ID. This ensures that the override is not applied to any unintended devices that share the same hardware or compatible ID as the intended target device.

Valid subkeys are **LocationPaths** and **ChildLocationPaths**. Note that while these subkeys are defined as optional, either the **LocationPaths** or **ChildLocationPaths** subkey must be specified under each **HardwareID** or **CompatibleID** subkey. Both subkeys can be specified if needed.

## LocationPaths Registry Key

| Registry key name | LocationPaths |
| --- | --- |
| **Required/optional** | Optional (**LocationPaths** and/or **ChildLocationPaths** must be present to indicate the parent/child relationship to which the device override applies) |
| **Format requirements** | None |
| **Valid subkeys** | * A string which defines the location path
* \* (asterisk)
 |

### Comments

The **LocationPaths** registry subkey indicates that only devnodes that have the specified HardwareID or CompatibleID value set on the devnode will be affected by the Removable device override value. Children of the specified devnode will not be affected by the override value, unless a **ChildLocationPaths** registry subkey is also specified.

## ChildLocationPaths Registry Key

| Registry key name | ChildLocationPaths |
| --- | --- |
| **Required/optional** | Optional (**LocationPaths** and/or **ChildLocationPaths** must be present to indicate the parent/child relationship to which the device override applies) |
| **Format requirements** | None |
| **Valid subseys** | * A string which defines the location path\* (asterix)
 |

### Comments

The **ChildLocationPaths** registry subkey indicates that only child devnodes of the parent devnode that have the specified HardwareID or CompatibleID value will be affected by the Removable device override value. The parent devnode will not be affected by the override value, unless a **LocationPaths** registry subkey is also specified or a **ChildLocationPaths** registry subkey is specified for its parent devnode.

## \* Registry Key

| Registry key name | \* |
| --- | --- |
| **Required/optional** | Optional (\* and/or a valid location path must be present to indicate the scope of the device override) |
| **Format requirements** | None |
| **Valid subkeys** | None |
| **Valid registry values under this key** | * Removable
 |

### Comments

The \* (asterisk) registry subkey can be used under the **LocationPaths** and/or **ChildLocationPaths** registry subkeys. The \* subkey indicates that the Removable device override value has global scope. The effect of the \* subkey is to apply the Removable device override value to all of the devnodes with the specified HardwareID or CompatibleID in accordance with the rules of the **LocationPaths** or **ChildLocationPaths** registry subkey under which it appears.

## “*Location Path*” Registry Key

| Registry key name | Valid “*Location Path*” value |
| --- | --- |
| **Required/optional** | Optional (\* and/or a valid location path must be present to indicate the scope of the device override) |
| **Format requirements** | None |
| **Valid subkeys** | None |
| **Example** | * PCIROOT(0)#PCI(1234)#USBROOT(0)#USB(1)#USB(3)#USB(3)
 |
| **Valid registry values under this key** | * Removable
 |

### Comments

Providing a valid location path registry subkey under a **LocationPaths** and/or **ChildLocationPaths** registry subkey will apply the Removable override value to only the device that exists at the specified location path. This allows the Removable device override value to be applied to a single instance of a device on the PC. Other devices with the same HardwareID or CompatibleID at other location paths will not be affected by such a Removable device override value.

By convention, the location path string takes the form *ServiceName(BusSpecificLocation)*. For example, PCI devices use PCI(*XXYY*), where *XX* is the device number, and *YY* is the function number. The string is unique to the device relative to its bus. The PnP Manager assembles the location path for each node in the devnode tree. Each devnode in the tree concatenates its service name string to the end of the location path string supplied by its parent devnode. Therefore, the position of any devnode in the tree can be uniquely identified using the location path.

The **Location Path** string for a given devnode can be found using Device Manager:

* Open Device Manager and locate the devnode on which the registry override is to be applied. In order to do this, you may need to change the view to **Devices by connection**.
* Right click on the devnode, click **Properties** and click the **Details** tab.
* In the Property drop down list, find the property **Location paths** property. This property will contain the location path string for this devnode and is the value that should be used for the device override registry key.

Note that it is possible that the devnode does not have a Location paths value. This is because the driver for this devnode or one of its parents does not implement the [GUID\_PNP\_LOCATION\_INTERFACE interface](http://msdn.microsoft.com/en-us/library/bb898834.aspx). In this case, check the parent devnode for a Location paths property.

The location paths override is intended to be used for overriding the Removable device capability of devices that are hardwired to a fixed bus location. This typically occurs in laptops, and includes the following devices:

* Wireless network adapters.
* Bluetooth adapters.
* Keyboards or pointing devices.

These devices exist on different internal buses at fixed locations that cannot be changed by the user. The Location paths override will allow one to specify that only the device at the given bus location will be affected by the ContainerID override. This prevents the override from affecting devices at other bus locations that may share the same HardwareID or CompatibleID as the override target. This is common when devices only specify a CompatibleIDto match an inbox driver.

When using a **ChildLocationPaths** registry subkey to override the Removable device capability of child devnodes enumerated by a parent devnode, it is often useful to only target child devnodes at specific locations, regardless of what kind of devices they are.

For example, a laptop may have an internal USB hub with both internal and external ports. If this USB hub is misreporting its internal ports as being external, any devices internally hardwired to these ports will be incorrectly recognized as being removable. Likewise, if all ports are misreported as being internal, any externally connected devices will be treated as if they are non-detachable parts of the laptop.

Discovering the location paths value for a device connected to an external USB port can be easily done by simply plugging any device into the port and observing its location paths property. Any other USB devices plugged into the same port should receive the same location paths value, because the parent bus and how it internally identifies a port will never change.

## Removable Registry Value

| Registry value name | *Removable* |
| --- | --- |
| **Required/optional** | Required |
| **Value type** | DWORD (32-bit) value |
| **Valid registry values** | * 0 – Indicates that the applicable devnodes should be regarded as not removable for the purpose of creating the ContainerID
* 1 – Indicates that the applicable devnodes should be regarded as removable for the purpose of creating the ContainerID
 |

### Comments

The Removable value must be created as a DWORD (32-bit) value type. Recognized values are {0, 1} as described above.

## Device Override Examples

The following examples illustrate how different removable overrides will affect the ContainerIDs that PnP creates for devices.

Figure . Key for the following examples

We will start with a hypothetical USB device connected to a PC. For this first example, we will assume that the device correctly reports its Removable device capability and there are no removable device overrides configured for it in the registry.

Figure . Devnode topology for a hypothetical PC and device

 shows the devnode topology for a PC and external device. As outlined previously, PnP examines the devnode topology to determine which devnodes belong to the PC and the device. In this case, PnP has assigned two ContainerIDs based on this topology. ContainerID {A} represents the PC, and all of the devnodes belonging to the PC receive this ContainerID. (An actual PC would have more devnodes than illustrated in this example.) ContainerID {B} represents the external device attached to the PC.

The following examples will demonstrate the effect of adding removable override registry entries to the device in .

### Example 1

The following shows a device override for a devnode matching a HardwareID at a specific Location Path. In this example the override will be applied to devnodes with HardwareID of USB\VID\_1234&PID\_5678 at the specified Location Path. The registry format is:

|  |
| --- |
| **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceOverrides****USB#VID\_1234&PID\_5678****LocationPaths****PCIROOT(0)#PCI(102)#** **USBROOT(0)#** **USB(1)** **Removable=0** |

This override will change the interpretation of the device topology by PnP. Notice that the devnode with HardwareID USB\VID\_1234&PID\_5678 has been marked as not removable in the registry. A new ContainerID will not be generated for this devnode, as PnP interprets the devnode as not being removable from its parent. Instead, USB\VID\_1234&PID\_5678 (and all of its children) will inherit the ContainerID of its parent. In this case, ContainerID {A} will be inherited as it is the first devnode above USB\VID\_1234&PID\_5678 which is assigned a ContainerID. The result of this override is a single device, as all of the devnodes in the tree have the identical ContainerID {A}. The device USB\VID\_1234&PID\_5678 is interpreted as being integrated with the PC.

This example illustrates a commonly encountered devnode topology: laptops with devices hardwired to specific bus locations which incorrectly report themselves as removable. Devices which are physically integrated with a PC, such as a webcam or biometric (fingerprint) reader, should not be reported as Removable because they cannot be physically separated from the PC by a user. The removable override allows an OEM to change how PnP interprets the removable capability, and thereby affect the ContainerID assignment for the device.

Figure . Override showing the effect of marking a devnode not removable

### Example 2

The following shows a device override for all devnodes matching a given HardwareID. In this example the override will be applied to devnodes with HardwareID of USB\VID\_062A&PID\_0000. The registry format is:

|  |
| --- |
| **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceOverrides****USB\VID\_062A&PID\_0000****LocationPaths****\*****Removable=1** |

This override will change the interpretation of the device topology in PnP. The device will be assigned two ContainerIDs, and therefore is seen by Windows as two devices. Notice that the devnode with HardwareID USB\VID\_062A&PID\_0000 is interpreted as removable for the purpose of grouping the devnodes into devices. The Removable capability flag on the PnP devnode is not changed.

Additionally, the \* registry subkey has been specified to indicate that this override should be applied to all devnodes on this PC with the HardwareID of USB\VID\_062A&PID\_0000.

Figure . Breaking a device into two device containers

### Example 3

The following shows the effect of the **ChildLocationPaths** device override key. In this case **ChildLocationPaths** will be applied to the USB\VID\_062A&PID\_0000 devnode. The registry format is:

|  |
| --- |
| **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceOverrides****USB#VID\_062A&PID\_0000****ChildLocationPaths****\*****Removable=1** |

The device will be assigned three ContainerIDs, and therefore is seen by Windows as three devices. Notice that the children of the parent devnode with HardwareID USB\VID\_062A&PID\_0000 are each interpreted as removable for the purpose of grouping the devnodes into devices. The PnP devnode property is not changed.

Additionally, the \* registry key has been specified which indicates that this override should be applied to the children of all devnodes on this PC with the parent HardwareID of USB\VID\_062A&PID\_0000.

 Figure . Device overrides using the ChildLocationPaths key

### Example 4

The following shows the effect of applying several of the device override keys covered in the previous examples:

|  |
| --- |
| **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceOverrides****USB#VID\_1234&PID\_5678** **LocationPaths****PCIROOT(0)#PCI(102)#USBROOT(0)#USB(1)****Removable=0****USB#VID\_062A&PID\_0000****ChildLocationPaths****\*****Removable=1****USB#VID\_062A&PID\_0002** **LocationPaths****\*****Removable=1** |

Figure . Several overrides applied together

# Verifying Correct Implementation of ContainerID

Verifying that a PC or device correctly implements ContainerID is an important part of the development process. An incorrect implementation will negatively impact the usability of the device.

Windows 7 introduces a new way for users to see and interact with devices: the Devices and Printers folder. The Devices and Printers folder presents the user with a view of all of the devices connected to the PC. Devices are displayed in the folder as they physically appear to the user, as a single “piece-of-plastic.”

IHVs and OEMs can customize their devices in the Devices and Printers folder in a variety of ways. A custom photorealistic icon and detailed information can be associated with the device and displayed to the user. Actions can be associated with the device, allowing device-related tasks to be launched. Together, these changes will enable users to interact with their devices in a consistent manner. Likewise, IHVs and OEMs will be able to easily expose device functionality.

To take advantage of the new capabilities provided by the Devices and Printers folder, devices must properly implement ContainerID as outlined in this document. The following section describes how proper implementation of ContainerIDs can be verified.

## Devices and Printers Folder

The simplest way to verify that a device conforms to the ContainerID requirements is to open the Devices and Printers folder to see how the device appears. If the device conforms to the ContainerID requirements outlined in this document, then there should be only one object in the Devices and Printers folder for that device. The following example illustrates this:



Figure . Devices and Printers folder showing USB mouse

In the example, the mouse is attached to a USB port on a laptop. This is the same mouse from , which has two devnodes associated with it. Notice that in the Devices and Printers folder only one instance of the mouse is shown, which matches the physical device. This device correctly implements the requirements for ContainerID. The physical device, for which there is one piece-of-plastic or “container,” is represented by one object in the Devices and Printers folder. In this example, the mouse does not contain a Microsoft OS ContainerID Descriptor or serial number. As a result, the ContainerID is being generated for the mouse using the Removable capability for the device.

If there is more than one instance of a device in the Devices and Printers folder when we expect only one, we would conclude that the device did not correctly implement the ContainerID requirements. One or more devnodes were improperly grouped into additional device containers for this device. In such a case, the following should be examined:

1. Is the Removable device capability set correctly for each devnode enumerated for the device?

This is the most common cause of multiple device instances in the Devices and Printers folder. Check that each devnode for the device has the Removable device capability set appropriately. The top-most devnode of the device should be reported as Removable, and children should be reported as not-Removable. Custom bus driver implementations need to correctly assign the Removable relationship for devnodes they enumerate.

Device Manager can be an invaluable tool when diagnosing these issues. By selecting **View > View by connection**, the complete devnode hierarchy can be examined. Locate the devnodes comprising your device in the devnode hierarchy. Open the **Properties** dialog for each devnode, click the **Details** tab and select the **Capabilities** entry in the Properties dropdown list. If the **Values** list box contains the **CM\_DEVCAP\_REMOVABLE** flag, then this devnode is marked as Removable, and PnP will create a new device container for this devnode and its non-Removable children.

1. Does the device contain a ContainerID or other unique identifier in the hardware?

Check that the format of the ContainerID or unique identifier in the hardware conforms to the format requirements for the given bus. These requirements are detailed in the section “.”

If devnodes for the device are enumerated by a custom bus driver, check that the bus driver properly implements the PnP minor IRP for device containers. Implementation details of this PnP minor IRP can be found in the section .”

1. Is the device simultaneously connected to the PC via more than one bus?

If the device is connected to the PC via two or more busses simultaneously, it is possible that two or more instances of the device may appear in the Devices and Printers folder. (One or more device instance for each bus to which the device is attached.) Solving this requires that the device report a ContainerID or a device-specific unique identifier, and that the value reported be identical on each bus.

# Call to Action

Device containers enable a new way to think about devices, with potential benefits to both users and the hardware industry. Users will benefit from the new Devices and Printers folder, enabling them to interact with their hardware in a more natural, intuitive way. The hardware industry will benefit from having the Devices and Printers folder as a launch point for new and exciting device-centric scenarios created by IHVs and OEMs.

To be successful, device manufacturers need to be aware of the device container concept and engineer their devices to take full advantage of this new functionality.

# Resources

**Windows 7 Device Experience**<http://www.microsoft.com/whdc/device/DeviceExperience/default.mspx>

**Microsoft OS Descriptors**
<http://www.microsoft.com/whdc/connect/usb/OS_Descdwn.mspx>

**PnP Device Capabilities**
<http://msdn.microsoft.com/en-us/library/aa491648.aspx>

**CM\_DEVCAP\_REMOVABLE**
<http://msdn.microsoft.com/en-us/library/ms792967.aspx>

**GUID\_PNP\_LOCATION\_INTERFACE Interface**<http://msdn.microsoft.com/en-us/library/bb898834.aspx>

**PnP-X: Plug and Play Extensions for Windows**
<http://www.microsoft.com/whdc/connect/rally/pnpx-spec.mspx>.

**UPnP Device Architecture, Version 1.0**
<http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20080424.pdf>

**Device Profile for Web Services**
<http://msdn.microsoft.com/webservices/webservices/understanding/specs/default.aspx?pull=/library/en-us/dnglobspec/html/devprofspecindex.asp>
<http://specs.xmlsoap.org/ws/2005/05/devprof/devicesprofile.pdf>

#### For the latest information about the Microsoft Windows family, see the [Windows Web site](http://www.microsoft.com/windows) at <http://www.microsoft.com/windows>.