Windows Hardware Compatibility Program

November 6, 2015 - **Microsoft makes no warranties, express or implied.**

Contents

Windows Hardware Compatibility Program

The new Windows Hardware Compatibility Program is an evolution of the Windows Logo Program and the Windows Hardware Certification Programs. It is designed to help your company deliver systems, software and hardware products that are compatible with Windows and run reliably on Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), and the next version of Windows Server.

Like previous versions, the new Windows Hardware Compatibility Program leverages the tests in the new Hardware Lab Kit (formerly the Hardware Certification Kit) to test your product. After passing certain tests, the program allows you to use the Windows logo as part of your marketing. End users trust the Windows logo as a sign of compatibility. Enterprise and government customers also look for the logo, or consult the Microsoft Certified Products list or the server catalog to see what systems, components and peripherals have been tested to ensure interoperability and reliability.

The new hardware compatibility program provides you with:

- Minimum engineering requirements that you can use to design your hardware.
- A subset of the tests in the new Hardware Lab Kit (formerly the Hardware Certification Kit) that focus on compatibility, interoperability and reliability.
- The opportunity to list your product on Microsoft's Certified Products List after passing the compatibility and reliability tests.
- Guidance for developing, testing and distributing drivers.
- Access to the Windows Dev Center hardware dashboard to manage submissions, track the performance of your device or app, review telemetry and much more.

New IHV Attested Signing Service

For Windows 10, Microsoft is creating a new service for IHVs who just need to get a driver signed for production. IHVs can go to the Windows Dev Center – Hardware Dashboard, when available, and request a signed driver by attesting to the quality. This process does not require HLK test results. Under this process the IHV would affirm the following in order to get a Microsoft signed driver:

- IHV attests that they have completely tested their driver
- IHV attests that their product and driver will not break interoperability
- IHV attests that they will monitor telemetry and remediate any issues (exact metrics to be determined at a later date)

When an IHV agrees to these terms, a signed driver will be returned. This signed driver can be distributed to end users, and eventually distributed via Windows Update. However, the driver cannot be used in a Compatibility Test System and the product will not be listed on the Certified Products List. The signature that is returned from this process is different than one that is returned after you submit HLK test results to Microsoft. However, functionally there is no difference between the two signatures.

This option is only available for Windows 10 for desktop editions operating system.

For more details about building your system, see:

- Minimum Hardware Specification - This specification defines the minimum hardware requirements necessary to:
  - Boot and run Windows 10.
  - Update and service Windows 10.
  - Provide a baseline user experience that is comparable with similar devices and computers.
- Windows Hardware Compatibility Specification - This specification defines the engineering requirements for passing the Windows Hardware Compatibility Program.

Official Test Playlist

The Windows Hardware Compatibility Program uses an official playlist to determine which devices meet the requirements for compatibility with Windows 10. All playlists that we have published are acceptable to use for submissions to the Hardware Developer Portal (sysdev).

The latest playlist can be downloaded at the following location:

- https://sysdev.microsoft.com/en-US/Hardware/CompatibilityPlaylists/

In this section:

Send comments about this topic to Microsoft

Windows Hardware Compatibility Program Requirements

The requirements define how to build Windows-compatible devices, systems, and filter drivers across all Windows Platforms. They were developed in collaboration with partners, and focus on ensuring compatibility, interoperability and reliability.

The requirements are validated by HLK tests and categorized as:

- System requirements
- Device and peripheral requirements for a stand-alone device
- Filter driver requirements Windows Filtering Platform drivers (WPF), file system filter drivers, antivirus, and Early Launch Anti-Malware (ELAM) filter drivers.

When products meet the minimum requirements it ensures that the application and device are compatible. Systems are required to use components which have also passed compatibility testing. Products submitted with passing results will continue to be included on the Certified Products List.

In this section

- Filter
- Components and peripherals
- Systems

Hardware Certification Policies and Processes

The WHCP Policies and Processes document below contains certification testing policies, product submission, and business requirements.

- WHCP Policies and Processes

Send comments about this topic to Microsoft

Filter

In this section:

- Filter.Driver.AntiVirus
- Filter.Driver.DeviceGuard
- Filter.Driver.EarlyLaunchAntiMalware
- Filter.Driver.FileSystem
- Filter.Driver.Fundamentals
- Filter.Driver.Network.LWF
- Filter.Driver.Security
- Filter.Driver.vSwitchExtension
- Filter.Driver.WindowsFilteringPlatform

Send comments about this topic to Microsoft

Filter.Driver.AntiVirus

Antivirus requirements for filter drivers

In this topic:

Filter.Driver.AntiVirus.Functionality

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows file systems, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows file systems, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Local File Systems
  - NT API, Win32 API and Win32 mapped IO API usage
  - Object ID functionality
  - Reparse points
  - Oplocks
  - System cache usage
  - Transactional capability
- Remote file systems
- Oplock semantics over SMB

Information about file system behavior: http://download.microsoft.com/download/4/3/8/43889780-8d45-4b2e-9d3a-c696a890309f/File%20System%20Behavior%20Overview.pdf

Information about Oplock semantics over SMB, see the [MS-SMB2] protocol document at: http://msdn.microsoft.com/en-us/library/cc246482(PROT.13).aspx

Filter.Driver.AntiVirus.IcarDetection

Anti-virus filter drivers must be architected to exercise basic anti-virus functionality, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Anti-virus filter drivers must be architected to exercise basic anti-virus functionality, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- File systems
- Anti-virus functionality

Filter.Driver.AntiVirus.MiniFilter

A file system filter driver must be a minifilter driver that uses the file systems filter manager.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement will be tested implicitly. The driver detection mechanism of the Windows Hardware Lab Kit will be written such that legacy file system filter drivers are not enumerated. Only minifilter drivers will be enumerated and surfaced in the kit. As such, a user will be unable to select a legacy filter driver for logo testing via the kit.

Information about filter manger and minifilter drivers available here:

http://msdn.microsoft.com/en-us/library/ff540402(v=VS.85).aspx

http://msdn.microsoft.com/en-us/windows/hardware/gg462968.aspx

Filter.Driver.AntiVirus.NamedPipeAndMailSlots

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Named Pipe and Mail Slots, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Named Pipe and Mail Slots, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Named Pipe file system
  - Functionality and stress for common APIs
  - Anonymous pipes
  - Pipe modes
  - Open modes
  - Invalid pipe names
  - Flushing pipe
  - Max pipe instance
  - Pipe direction (in/out/duplex)
  - Input and output buffer sizes
  - Various call semantics, such as reconnecting a pipe that has been disconnected at the server end.
  - Behavior validation of all named pipes operations for each distinct state of a pipe instance.
  - Performance for named pipe creation and connection.
  - Throughput for different in/out buffer sizes and number of clients.
  - Scalability of increasing number of clients to time it takes for a connection to a named pipe instance

- Mail Slot file system
- Functionality and stress for common APIs

Information about Named Pipe and Mail Slots can be found at:

http://msdn.microsoft.com/en-us/library/aa365574(v=VS.85).aspx

Filter.Driver.AntiVirus.RegistryAndProcess

Kernel mode filter drivers must be architected to maximize the reliability and functionality of the Windows registry and processes, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel mode filter drivers must be architected to maximize the reliability and functionality of the Windows registry and processes, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Registry
  - NT API and Win32 API usage
  - Key functions
  - Transaction registry operations
  - Symbolic link behavior
- Process
  - General module management
  - Race conditions at thread/process termination
  - Process management callback functionality
- Thread and process handle operations

Filter.Driver.AntiVirus.Winsock

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows Sockets, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows Sockets, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Winsock
- Winsock API functionality

Information about Winsock APIs can be found at:

http://msdn.microsoft.com/en-us/library/ms740673(VS.85).aspx

Send comments about this topic to Microsoft

Filter.Driver.DeviceGuard

All kernel drivers must be built to be compatible with Device Guard.

In this topic:

Filter.Driver.DeviceGuard.DriverCompatibility

Applies to	Windows 10 x64 Windows Server 2016 Technical Preview x64

Description

Windows 10 has a new feature called Device Guard that gives organizations the ability to lock down devices in a way that provides advanced malware protection against new and unknown malware variants as well as Advanced Persistent Threats (APTs). Device Guard can use hardware technology and virtualization to isolate the Code Integrity (CI) decision-making function from the rest of the Windows operating system. When using virtualization-based security to isolate Code Integrity, the only way kernel memory can become executable is through a Code Integrity verification. This means that kernel memory pages can never be Writable and Executable (W+X) and executable code cannot be directly modified.

Details are available in the Windows Hardware Certification blog.

Send comments about this topic to Microsoft

Filter.Driver.EarlyLaunchAntiMalware

This section describes requirements for early launch driver.

In this topic:

Filter.Driver.EarlyLaunchAntiMalware.BackupDriver

Early launch anti-malware drivers must include a backup copy in case of corruption.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The anti-malware (AM) driver is critical to the boot success of the computer. If the driver gets corrupted, then the boot may not succeed. To provide the best user experience, it is required that when the AM driver is installed, it also installs a copy in the driver backup store. This ensures a smooth remediation experience in the case that the primary driver gets corrupted.

The location of the ELAM backup store is defined by Windows, and stored in the registry:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\EarlyLaunch ! BackupPath

Design Notes:

The early launch anti-malware (AM) drivers are started soon after the NTOS kernel starts. For each subsequent boot driver, the AM driver receives a callback from the PnP manager to determine whether the boot driver should be initialized. The AM driver evaluates the boot driver and must return good, bad, or unknown. Based on the returned classification and defined policy, the PnP manager decides whether to initialize the boot driver.

Filter.Driver.EarlyLaunchAntiMalware.ELAMSignatureAttributes

Requirement for the SignatureAttribute section in the ELAM INF files

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

ELAM files must be signed with a special signature as part of the submission process. The process of determining what signature each module needs is being standardized; each INF file must now include a SignatureAttributes section that uniquely identifies what type of signature is applicable for the associated driver binaries. Adding this section to existing inf files is a very simple process.

An example follows:

```
[SignatureAttributes]
ELAMFILE.dll = SignatureAttributes.Elam
[SignatureAttributes.Elam]
Elam=true
```

Filter.Driver.EarlyLaunchAntiMalware.MVIMembership

Early launch anti-malware drivers may only be created by MVI members.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Any early launch anti-malware (AM) driver may only be created by Microsoft Virus Initiative (MVI) members.

Design Notes:

The early launch AM drivers are started soon after the NTOS kernel starts. For each subsequent boot driver, the AM driver receives a callback from the PnP manager to determine whether the boot driver should be initialized. The AM driver evaluates the boot driver and must return good, bad, or unknown. Based on the returned classification and defined policy, the PnP manager decides whether to initialize the boot driver.

Filter.Driver.EarlyLaunchAntiMalware.Performance

Early launch anti-malware drivers must be performant.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Callback Latency:

The anti-malware (AM) driver is required to return a result for each callback within 0.5 ms of receiving the callback.

Memory Allocation:

The AM driver, including both the driver image as well as its configuration (signature) data, is required to have a limited memory footprint of 256 KB or less.

Unload Blocking:

Each AM driver will receive a synchronous callback after the last boot driver has been initialized indicating that the AM driver will be unloaded. At this point, the AM driver must clean up and save any persistent status information. This must occur within 0.5 ms as measured from the time when the kernel issues the callback to the driver to the time that the AM driver returns the callback.

Design Notes:

The early launch AM drivers are started soon after the NTOS kernel starts. For each subsequent boot driver, the AM driver receives a callback from the PnP manager to determine whether the boot driver should be initialized. The AM driver evaluates the boot driver and must return good, bad, or unknown. Based on the returned classification and defined policy, the PnP manager decides whether to initialize the boot driver.

Filter.Driver.EarlyLaunchAntiMalware.SignatureData

Early launch anti-malware drivers must only use signature data stored in the Microsoft-specific location.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The AM driver must get its malware signature data from a single, well-known location and no other. The signature data shall be stored in the registry in a new "ELAM" hive under HKLM that is loaded by Winload, and will therefore be available to the AM driver prior to the file system being initialized. Each AM driver will have a unique key in which to store their signature blob. The registry path and key shall be of the format HKLM\ELAM\<Vendor Name>\Measured : Binary = <blob>.

Design Notes:

The early launch anti-malware (AM) drivers are started soon after the NTOS kernel starts. For each subsequent boot driver, the AM driver receives a callback from the PnP manager to determine whether the boot driver should be initialized. The AM driver evaluates the boot driver and must return good, bad, or unknown. Based on the returned classification and defined policy, the PnP manager decides whether to initialize the boot driver.

Send comments about this topic to Microsoft

Filter.Driver.FileSystem

In this topic:

Filter.Driver.FileSystem.Functionality

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows file systems, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows file systems, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Local file systems
  - NT API, Win32 API and Win32 mapped IO API usage
  - Object ID functionality
  - Reparse points
  - Oplocks
  - System cache usage

- Transactional capability

- Remote file systems

- Oplock semantics over SMB

Information about file system behavior: http://download.microsoft.com/download/4/3/8/43889780-8d45-4b2e-9d3a-c696a890309f/File%20System%20Behavior%20Overview.pdf

Information about Oplock semantics over SMB, see the [MS-SMB2] protocol document at: http://msdn.microsoft.com/en-us/library/cc246482(PROT.13).aspx

Filter.Driver.FileSystem.MiniFilter

A file system filter driver must be a minifilter driver using the file systems Filter Manager.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement will be tested implicitly. The gatherer will be written such that it enumerates and surfaces only minifilter drivers for the HLK. Hence, a user will be unable to select a legacy filter driver for certification testing.

Information about Filter Manger and minifilter drivers available here:

http://msdn.microsoft.com/en-us/library/ff540402(v=VS.85).aspx

Filter.Driver.FileSystem.NamedPipeAndMailSlots

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Named Pipe and Mail Slots, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel Mode filter drivers must be architected to maximize the reliability and functionality of Named Pipe and Mail Slots, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Named Pipe file system

  - Functionality and stress for common APIs

- Anonymous pipes
- Pipe modes
- Open modes
- Invalid pipe names
- Flushing pipe
- Max pipe instance
- Pipe direction (in/out/duplex)
- Input and output buffer sizes
- Various call semantics, such as reconnecting a pipe that has been disconnected at the server end.
- Behavior validation of all named pipes operations for each distinct state of a pipe instance.
- Performance for named pipe creation and connection.
- Throughput for different in/out buffer sizes and number of clients.
- Scalability of increasing number of clients to time it takes for connection to a named pipe instance

- Mail Slot file system
- Functionality and stress for common APIs

Information about Named Pipe and Mail Slots can be found at:

http://msdn.microsoft.com/en-us/library/aa365574(v=VS.85).aspx

Filter.Driver.FileSystem.RegistryAndProcess

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows registry and processes, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel mode filter drivers must be architected to maximize the reliability and functionality of Windows registry and processes, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Registry
  - NT API and Win32 API usage
  - Key functions
  - Transaction registry operations
  - Symbolic link behavior
- Process
  - General module management
  - Race conditions at thread/process termination
  - Process management callback functionality
- Thread and process handle operations

Send comments about this topic to Microsoft

Filter.Driver.Fundamentals

Corresponds to device driver fundamentals, but for filter drivers

In this topic:

Filter.Driver.Fundamentals.DriverQuality

A filter driver must be of high quality.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Driver components must not cause the system to crash or leak resources. These resources include, but are not limited to the following:

- Memory
- Graphics Device Interface (GDI) or user objects
- Kernel objects such as files, mutex, semaphore, and device handles
- Critical sections
- Disk space
- Printer handles

Design Notes:

Sleep & PNP with IO Before And After Test - Test cycles the system through all sleep states and does basic PNP on all devices on the system.

This test will be run with Driver Verifier enabled with standard settings.

Send comments about this topic to Microsoft

Filter.Driver.Network.LWF

LAN requirements

In this topic:

Filter.Driver.Network.LWF.Base

All light weight filters must be NDIS 6.30 or greater.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All light weight filters must be NDIS 6.30 or greater and be compliant to the NDIS specification on MSDN.

Filter.Driver.Network.LWF.MTUSize

All light weight filters must be able to accept arbitrary packet sizes which might be greater than the miniport's MTU.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All light weight filters must be NDIS 6.30 or greater. All light weight filters must be able to accept arbitrary packet sizes which might be greater than the miniport's MTU.

Send comments about this topic to Microsoft

Filter.Driver.Security

Additional TDI filter driver and LSP requirements related to security

In this topic:

Filter.Driver.Security.NoTDIFilterAndLSP

No TDI filters or LSPs are installed by the driver or associated software packages during installation or usage.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

There can be no use of TDI filters or LSPs by either kernel mode software or drivers, or user mode software or drivers.

Send comments about this topic to Microsoft

Filter.Driver.vSwitchExtension

In this topic:

Filter.Driver.vSwitchExtension.ExtensionRequirements

Filter drivers that implement VM Switch Extensibility must support required functionalities, modes, and protocols.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Filter drivers that implement VM Switch Extensibility must support required functionalities, modes, and protocols.

Requirements

- An extension must pass NDIS Filter logo requirements.
  - An extension must have a valid INF.
  - An extension must make only NDIS, WDF, or WDM calls; any calls to other kernel mode components are not allowed.
- An extension must support Hyper-V Live Migration.
  - Don't break LM, Save/Restore, Export/Import
  - Don't block saved data from another extension
  - Don't block other extension interactions
- All traffic passing through a virtual switch coming from a VM, a host VNIC, external NIC, or extension must not have headers modified by extensions. Exceptions from this requirement include:
  - Redirecting traffic to a network appliance
  - Mutable IP header fields (as specified in RFC 4302 section 3.3.3.1.1.1 for IPv4 and section 3.3.3.1.2.1 for IPv6)

- An unconfigured extension must not "break" connectivity between the host and external network.
- A capture extension must not "break" connectivity between vSwitch ports.
- An extension must pass the following switch/port/NIC configuration OIDs down the stack of an extension:
  - OID_SWITCH_PARAMETERS
  - OID_SWITCH_PORT_ARRAY
  - OID_SWITCH_PORT_TEARDOWN
  - OID_SWITCH_PORT_DELETE
  - OID_SWITCH_NIC_ARRAY
  - OID_SWITCH_NIC_CONNECT
  - OID_SWITCH_NIC_DISCONNECT
  - OID_SWITCH_NIC_DELETE
  - OID_SWITCH_NIC_REQUEST
- An extension must pass the following policy/status OIDs that it does not consume down the stack:
  - OID_SWITCH_PORT_PROPERTY_ADD
  - OID_SWITCH_PORT_PROPERTY_UPDATE
  - OID_SWITCH_PORT_PROPERTY_DELETE
  - OID_SWITCH_PROPERTY_ADD
  - OID_SWITCH_PROPERTY_UPDATE
  - OID_SWITCH_PROPERTY_DELETE
  - OID_SWITCH_PORT_FEATURE_STATUS_QUERY
  - OID_SWITCH_FEATURE_STATUS_QUERY
- An extension must pass the following policy OIDs down the stack:
  - OID_SWITCH_PORT_PROPERTY_ENUM

    - OID_SWITCH_PROPERTY_ENUM
- An extension must pass the following up the stack:
    - NDIS_SWITCH_NIC_STATUS_INDICATION
- A "capture" extension must not call any of the following functions:

Design Notes:

See the VM Switch Extensibility Specification.

Send comments about this topic to Microsoft

Filter.Driver.WindowsFilteringPlatform

In this topic:

- Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.StreamInjection.NoStreamStarvation
- Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.SupportPowerManagedStates
- Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.WFPObjectACLs
- Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.Winsock
- Filter.Driver.WindowsFilteringPlatform.Firewall.DisableWindowsFirewallProperly
- Filter.Driver.WindowsFilteringPlatform.Firewall.NotOnlyPermitAllFilters
- Filter.Driver.WindowsFilteringPlatform.Firewall.Support5TupleExceptions
- Filter.Driver.WindowsFilteringPlatform.Firewall.SupportApplicationExceptions
- Filter.Driver.WindowsFilteringPlatform.Firewall.SupportMACAddressExceptions
- Filter.Driver.WindowsFilteringPlatform.Firewall.UseWindowsFilteringPlatform
- Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportAddressResolution
- Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportDynamicAddressing
- Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportIPv4
- Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportIPv6
- Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportNameResolution
- Filter.Driver.WindowsFilteringPlatform.Scenario.Support6to4
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportAutomaticUpdates
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportBasicWebsiteBrowsing
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportFileAndPrinterSharing
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportICMPErrorMessages
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportInternetStreaming
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportMediaExtenderStreaming
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportMobileBroadBand
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportPeerNameResolution
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportRemoteAssistance
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportRemoteDesktop
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportTeredo
- Filter.Driver.WindowsFilteringPlatform.Scenario.SupportVirtualPrivateNetworking
- Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.InteropWithOtherExtensions
- Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.NoEgressModification

- Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.SupportLiveMigration
- Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.SupportRemoval
- Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.SupportReordering

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.AppContainers.SupportModernApplications

WFP-based products must not block App Container apps that are operating within their declared network intentions by default, and should only block App Container apps when following specific user/admin intention or protecting the system against a specific threat.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products must not block App Container apps that are operating within their declared network intentions by default, and should only block App Container apps when following specific user/admin intention or protecting the system against a specific threat.

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.CleanUninstall

WFP-based products must stop cleanly and clean up all running state upon uninstall.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is to ensure that host firewalls do not leave unused objects upon uninstall, thereby potentially causing diagnostic issues if another separate host firewall is installed on the same PC.

The following WFP objects need to be cleaned up: Provider, providerContext, Filter, subLayer, or callout.

In addition, additional installation requirements for applications (via the Software logo program) must be met.

Design Notes:

Applications can use either an MSI, or another installer that meets this requirement to ensure a satisfactory install/uninstall experience on a Windows® based PC.

The installation requirements for applications (in the Software Logo Program) are located in the following link:

http://www.microsoft.com/downloads/details.aspx?FamilyID=27028822-B172-4CEC-91A3-26B610A4DA79&displaylang=en

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.ConnectionProxying.NoDeadlocks

WFP-based products that redirect or proxy at redirect layers (connect redirect), must use the new proxying API so that other WFP-based products can determine that the connection has been proxied.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products which redirect or proxy at redirect layers (connect redirect), must use the new proxy'ing API so that other WFP-based products can determine that the connection has been proxy'ed.

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.FwpmFilters.MaintainOneTerminating

WFP-based products must create and maintain at least one terminating FWPM_FILTER object.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A terminating filter is one that returns a permit / block decision. It may exist as a static filter or within a callout. The intent behind this requirement is to ensure that premium host firewalls perform at least one permit or block decision and not simply maintain filters only for inspection purposes, whereas basic host firewalls may do so through WFP or through other means such as TDI, NDIS, and WinSock LSP filters.

Design Notes:

The definition for the FWPM_FILTER object can be found in the following URL:
http://go.microsoft.com/fwlink/p/?linkid=116902&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.FwpmProviders.AssociateWithObjects

WFP-based products must associate all of their provider contexts, filters, sublayers, and callouts with their corresponding identifying provider object.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

For examples that illustrate the code behavior expected for various types of objects, please see below:

Reference the name & product of the company within an identifying provider object:

```
const PWSTR pCompanyName = L"Microsoft Corporation";
const PWSTR pProductName = L"Windows Firewall";
FWPM_PROVIDER0 myProvider;
myProvider.displayData.name = pCompanyName;
myProvider.displayData.description = pProductName;
```

Initialize the provider object:

```
FWPM_PROVIDER_CONTEXT0 myProviderContext;
FWPM_PROVIDER0 myProvider;
myProviderContext.providerKey = &(myProvider.providerKey);
```

Initialize the subLayer object & associate it to your respective provider object:

```
FWPM_SUBLAYER0 mySubLayer;
FWPM_PROVIDER0 myProvider;
mySubLayer.providerKey = &(myProvider.providerKey);
```

Initialize the callout object & associate it to your respective provider object:

```
FWPM_CALLOUT0 myCallout;
FWPM_PROVIDER0 myProvider;
myCallout.providerKey = &(myProvider.providerKey);
```

Initialize the filter object & associate it to your respective provider object:

```
FWPM_FILTER0 myFilter;
FWPM_PROVIDER0 myProvider;
myFilter.providerKey = &(myProvider.providerKey);
```

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.FwpmProviders.MaintainIdentifying

WFP-based products must create and maintain at least one identifying FWPM_PROVIDER provider object.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

An "identifying provider object" **must** reference the name & product of the company as shown in the example below.

FWPM_PROVIDER0

- All vendors must create and maintain at least 1 provider.
- The provider.displayData.Name must contain the name of the company.
- The provider.displayData.Description must contain the name of the product.

All objects created & "owned" by the vendor must reference only their provider(s):

```
const PWSTR pCompanyName = L"Microsoft Corporation";
const PWSTR pProductName = L"Windows Firewall";
FWPM_PROVIDER0 myProvider;

myProvider.displayData.name = pCompanyName;
myProvider.displayData.description = pProductName;
```

Design Notes:

The definition of the FWPM_PROVIDER object can be found in the following URL:

http://go.microsoft.com/fwlink/p/?linkid=116844&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.FwpmSublayers.UseOwnOrBuiltIn

WFP-based products must use only their own sublayer or one of the built-in sublayers.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A host firewall's own sublayer may be used to ensure that its filters must not be bypassed by a higher weight filter from another host firewall. In addition, a host firewall must not override filters belonging to another host firewall.

Design Notes:

The definition for the FWPM_SUBLAYERobject can be found in the following URL:

http://go.microsoft.com/fwlink/p/?linkid=116845&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.NetworkDiagnosticsFramework.HelperClass

WFP-based products must include a Network Diagnostics Framework (NDF) helper class that extends the Filtering Platform helper class (FPHC).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The Windows Filtering Platform (WFP) includes a Network Diagnostics Framework (NDF) helper class, called the Filtering Platform helper class (FPHC). FPHC can help to identify the root causes of connectivity issues caused by WFP. A host firewall can invoke its own NDF helper class. FPHC extensibility allows these third-party helper classes to be invoked during diagnostics.

FPHC can identify WFP as the cause of a connectivity issue. If available, FPHC can also identify the provider that created the filter that is blocking network traffic. FPHC passes this information to NDF, which in turn can then notify the user that WFP is causing the connectivity problem and give the name of the provider blocking traffic.

However, the FPHC cannot suggest a corrective action to the user, nor can it provide the reason that the filter is blocking traffic to the user. Only an FPHC extension can perform those tasks.

Host firewalls must be able to successfully diagnose the inbound/outbound connection failures caused by the host firewall, and provide an appropriate response to the end-user based on the diagnosis. (eg. Repair mechanism, message explaining to the user the reason why the connection failed, etc).

Design Notes:

More information regarding NDF and FPHC can be found in the following links:

NDF : http://go.microsoft.com/fwlink/p/?linkid=125463&clcid=0x409

FPHC : http://go.microsoft.com/fwlink/p/?linkid=125464&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.NoAccessViolations

WFP-based products must not be the resulting cause of any access violation under high load or during driver load/unload.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products must not be the resulting cause of any access violation under high load or during driver load/unload (while under network load or not).

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.NoTamperingWith3rdPartyObjects

WFP-based products must not attempt to remove or alter another WFP-based product's WFP objects and built-in objects.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This ensures interoperability between multiple host firewalls' WFP objects within the operating system.

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.PacketInjection.NoDeadlocks

WFP-based products must not continually modify network packets that have already been modified and re-injected, so as to create potential deadlocks.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Firewalls may use callouts to modify and re-inject network packets, when filtering at any layer. One or many host firewalls may be present on the same system. When there is only one host firewall is present on the system, continually modifying & re-injecting the same packets may result in reduced performance and is to be avoided. When multiple host firewalls (with callouts) are present on the system, the same network packet(s) may continually be modified by multiple callouts. When a host firewall continually modifies and reinjects the same packet, it may result in the network packet never getting processed and could potentially create a deadlock, which is to be avoided.

Host firewalls must not modify and reinject the same network packet more than 2 times per layer. If such a situation occurs, host firewalls may choose to let the packet go through, or drop the network packet.

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.StreamInjection.NoStream Starvation

WFP-based product callouts at FWPM_LAYER_STREAM must not starve the data throughput.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To "Not Starve" means that Stream layer callout indications should not be pended to queue up more than 8 MB of data.

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.SupportPowerManagedStates

WFP-based products must ensure network connectivity upon recovering from power managed states.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Tests must be run on a machine that supports all the power states (standby, hibernate, hybrid, shutdown, restart). Host Firewalls allow the system to enter into and recover from the above mentioned power managed states. Upon resuming from those particular power managed states, requirements from WFP should be met.

Firewalls should never pend packets such that a power state change refuses to work due to the pended packets.

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.WFPObjectACLs

WFP-based products must ACL all of their objects in a way that any other WFP-based product can at least enumerate those objects using the corresponding WFP enumeration APIs.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products must ACL all of their objects in a way that any other WFP-based product can at least enumerate those objects using the corresponding WFP enumeration APIs.

This is to make sure that all WFP objects on the system can be enumerated by any Host firewall or application for diagnostic purposes.

Design Notes:

As an example, Filter objects must be able to be enumerated by the FwpmFilterEnum function documented in the following URL:

http://go.microsoft.com/fwlink/p/?linkid=116839&clcid=0x409

Similarly, enumeration functions for other objects (provider, sublayer etc) can be found in the following URL: http://go.microsoft.com/fwlink/p/?linkid=116840&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.ArchitecturalDesign.Winsock

Kernel mode filter drivers are architected to maximize the reliability and functionality of Windows Sockets, as well as interact accurately with the core components of the operating system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel mode filter drivers are architected to maximize the reliability and functionality of Windows Sockets, as well as interact accurately with the core components of the operating system. Some areas of particular interest are:

- Winsock
- Winsock API functionality

Information about Winsock APIs can be found at:

http://msdn.microsoft.com/en-us/library/ms740673(VS.85).aspx

Filter.Driver.WindowsFilteringPlatform.Firewall.DisableWindowsFirewallProperly

Host firewalls must disable the Windows firewall using only the supported method.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Host firewalls are provided with the ability to selectively turn parts of Windows Firewall on or off. These parts specify different types of rules (and subsequently filter sets), and may also be referred to as categories. Filter sets that may be selectively turned off are Boot-Time Filters, Firewall Filters, Connection Security Filters, and Stealth Filters.

The 'Register' interface is supported by the HNetCfg.FwProducts COM object. The put_DisplayName() call must be used to fill in your product information.

Before turning off the firewall rules category, vendor firewalls must ensure that all filters must be installed.

This requirement ensures better interoperability with Windows. In addition, if all installed host firewalls on the system are uninstalled for any reason, Windows Firewall is aware of this, and will automatically turn on the firewall filters, ensuring that the system is always protected.

The Connection Security filters need to remain enabled to keep Windows scenarios protected. Specifically, the Connection Security filters ensure that the system supports communications that require authentication and encryption.

Design Notes:

This requirement ensures that firewall vendors disable Windows Firewall per documented guidelines.

Filter.Driver.WindowsFilteringPlatform.Firewall.NotOnlyPermitAllFilters

Host firewalls must not have only "permit_all" filters.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Host firewalls must not circumvent the intent of the Windows Filtering Platform API tests, by simply maintaining all 'permit_all' filters for all kinds of network traffic, which essentially is not meaningful filtering of network traffic. This applies to both, static as well as callout filters. Similarly, Host firewalls

must not maintain only 'block_all' filters. However, that will be addressed when testing for consumer scenarios.

Filter.Driver.WindowsFilteringPlatform.Firewall.Support5TupleExceptions

All host based firewalls must be able to block/allow by 5-tuple parts (including port (ICMP type and code, UDP and TCP) IP address, and protocol (e.g. UDP/TCP/ICMP)).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All host based firewalls must be able to block/allow by 5-tuple parts (including port (ICMP type and code, UDP and TCP) IP address, protocol (e.g. UDP/TCP/ICMP)).

Filter.Driver.WindowsFilteringPlatform.Firewall.SupportApplicationExceptions

WFP-based products must support exceptions from corresponding applications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

In addition to supporting scenarios based on applications within Windows® it is important to support applications (installed by the home user), that are registered with the host firewall for filtering purposes. Firewalls may use parameters such as path, ports, etc as basis to permit or block application specific traffic. This scenario will need to work with native IPv4, native IPv6, 6to4, and Teredo packets.

The word 'support' refers to the host firewall's capability to ensure exceptions from applications work with the host firewall, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Filter.Driver.WindowsFilteringPlatform.Firewall.SupportMACAddressExceptions

All host based firewalls that have filters in L2 (Native/Mac) layers must be able to Block or Allow by MAC address.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All host based firewalls that have filters in L2 (Native/Mac) layers must be able to Block or Allow by MAC address.

Filter.Driver.WindowsFilteringPlatform.Firewall.UseWindowsFilteringPlatform

Firewalls must comply with Windows Filtering Platform based APIs for filtering network traffic on home user solutions.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

There must be no TDI, NDIS, WinSock LSP filters present upon installation of the host firewall on the PC. Only Windows Filtering Platform (WFP) based static filters / callouts must be used on home user products.

Design Notes:

For more information on Windows Filtering Platform, please see the following link:
http://go.microsoft.com/fwlink/p/?linkid=116899&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportAddressResolution

WFP-based products must support allowing for successful ARP and ICMP Neighbor Discovery exchanges.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products must support ARP (for IPv4) and ICMP Neighbor Discovery (for IPv6) exchanges.

Firewalls allow the system to send out ARP and ICMP Neighbor Discovery requests and replies, as well as receive ARP and ICMP Neighbor Discovery requests and replies.

The word 'support' refers to the host firewall's capability to make ARP work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

Host firewalls should allow the PC to send out ARP requests on behalf of another node rather than only on behalf of itself, when ICS is running on the host.

As part of Internet Connection Sharing's (ICS) DHCP functionality, ICS DHCP can send out ARP requests on behalf of another node in the subnet.

Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportDynamicAddressing

WFP-based products support allowing for successful DHCP exchanges over both IPv4 and IPv6.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Host firewalls support allowing successful DHCP exchanges over both IPv4 and IPv6.

DHCP DISCOVER, DHCP REQUEST & DHCP INFORM packets can be transmitted over outbound UDP source port 68 to destination port 67. DHCP OFFER & DHCP ACK & DHCP NACK packets can be received over inbound UDP source port 67 to destination port 68. DHCPv6 packets can be transmitted over outbound UDP source port 546 to destination port 547. DHCPv6 packets can be received over Inbound UDP Source Port 547 to destination port 546.

The word 'support' refers to the host firewall's capability to allow successful DHCP exchanges, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc. to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

Details can be found in the following URL:
http://go.microsoft.com/fwlink/p/?linkid=116834&clcid=0x409

Host firewalls should allow DHCP inbound and outbound as the server over the wireless interface when a service like ICS is running on the host.

Internet Connection Sharing (ICS) acts as a DHCP server and expects to receive incoming DHCP clients.

DHCP DISCOVER, DHCP REQUEST & DHCP INFORM packets can be received over Inbound UDP source port 68 to destination port 67.

DHCP OFFER & DHCP ACK & DHCP NACK packets can be transmitted over outbound UDP source port 67 to destination port 68.

Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportIPv4

WFP-based products must support IPv4 traffic.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is to ensure that consumer host firewalls or other filtering components do not cause the loss of basic IPv4 connectivity on the PC.

The word 'support' refers to the host firewall's capability to make IPv4 work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

More information about IPv4, RFCs can be found in the following link:
http://go.microsoft.com/fwlink/p/?linkid=116835&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportIPv6

WFP-based products must support IPv6 traffic.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows® has IPv6 enabled by default. Host firewalls should not break native IPv6 connectivity (and therefore, Windows scenarios based on IPv6) for customers.

The word 'support' refers to the host firewall's capability to make IPv6 work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

More information about IPv6 can be found in the following link:
http://go.microsoft.com/fwlink/p/?linkid=116832&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.NetworkingFundamental.SupportNameResolution

WFP-based products must support allowing for successful DNS client queries.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

DNS QUERY packet can be sent out over [Outbound UDP Destination Port 53 (Domain Name Server)] and DNS QUERY RESPONSE packet to be received over [Inbound UDP Source Port 53 (Domain Name Server)]. Host firewalls should allow successful DNS client queries over both IPv4 and IPv6.

The word 'support' refers to the host firewall's capability to allow successful DNS client queries, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

More information about DNS, RFCs can be found in the following link:
http://go.microsoft.com/fwlink/p/?linkid=116835&clcid=0x409

Host firewalls should allow this type of DNS traffic (Host as a server) over the wireless interface when a service like ICS is running on the host.

This requirement applies to Internet Connection Sharing that acts as a DNS server (proxy) and expects receiving incoming DNS requests from clients on destination UDP port 53, and respond to the DNS client with DNS response with destination UDP port 53.

Filter.Driver.WindowsFilteringPlatform.Scenario.Support6to4

WFP-based products must support 6to4.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

In certain markets, 6to4 technologies may help certain customers move to IPv6 connectivity. The following guidelines may help meet this requirement:

- Host firewalls allow for the system to send and receive IPv6 packets over IPv4 protocol 41.

The word 'support' refers to the host firewall's capability to 6to4 work, if the application/user/network needs it. The host firewall must also have properly configured objects such

as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

Please refer to the following article below for further information on 6to4:

http://go.microsoft.com/fwlink/p/?linkid=116837&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportAutomaticUpdates

WFP-based products must support Automatic Updates in Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is related to Automatic Updates / Windows Update (WU), which is a key scenario through which important patches are installed on your PC to keep it up to date. The following guideline may help meet this requirement:

- Host firewalls allow outbound TCP connections to destination ports 80 & 443.

The word 'support' refers to the host firewall's capability to make Automatic Updates work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

For more information on Windows Updates/ Automatic Updates, please see the following link:
http://go.microsoft.com/fwlink/p/?linkid=116898&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportBasicWebsiteBrowsing

WFP-based products must support basic internet browsing experiences.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is to ensure that basic internet browsing experiences are supported upon installation of a host firewall on a Windows® based computer.

Host firewalls must allow TCP packets over Ports 80 and 443 to support this scenario. This scenario must work with native IPv4, native IPv6, 6to4, and Teredo packets.

The word 'support' refers to the host firewall's capability to ensure a successful internet browsing experience, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportFileAndPrinterSharing

WFP-based products must support file and printer sharing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is to ensure that home users will be able to share content to and from other PCs inside of their home network, in addition to printing content on shared printers.

Host firewalls must allow UDP packets specific to protocol 17 over Ports 137 / 138, and TCP packets specific to protocol 6 over ports 139/445. This scenario must work with native IPv4, native IPv6, 6to4, and Teredo packets.

TCP packets should be allowed over ports 5357/5358 & UDP packets should be allowed over port 3702. This scenario should work with native IPv4, native IPv6, 6to4 and Teredo packets.

The word 'support' refers to the host firewall's capability to make file and printer sharing work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

Please refer to the following link for more information:

http://go.microsoft.com/fwlink/p/?linkid=116838&clcid=0x409

Please refer to the following documents for more information:

- HomeGroup Firewall Requirements: http://technet.microsoft.com/en-us/appcompat/default.aspx
- Network Location Dialog: http://technet.microsoft.com/en-us/appcompat/default.aspx
- PNRP: http://technet.microsoft.com/en-us/appcompat/default.aspx

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportICMPErrorMessages

WFP-based products must support ICMP error messages and discovery functions

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is to ensure that host firewalls support ICMP error messages (per IETF RFCs 4890 and RFC 2979), for inbound/outbound packets that must not be dropped. Important discovery functions must also be supported. The specific error messages that need to be supported for both ICMPv4 and ICMPv6 are: Destination Unreachable, Time Exceeded and Parameter Problem. In addition, for ICMPv6, Packet too big, Router solicitation, Neighbor solicitation, Router advertisement, and neighbor advertisement discovery functions must be supported.

The word 'support' refers to the host firewall's capability to make ICMP work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

For more information, please see http://go.microsoft.com/fwlink/p/?linkid=116835&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportInternetStreaming

WFP-based products must support Internet streaming and Media sharing for media player network sharing services.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is related to Automatic Updates / Windows Update (WU), which is a key scenario through which important patches are installed on your PC to keep it up to date. The following guidelines may help meet this requirement:

Host firewalls allow outbound TCP connections to destination ports 80 & 443.

The word 'support' refers to the host firewall's capability to make Automatic Updates work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

For more information on Windows Updates/ Automatic Updates, please see the following link: http://go.microsoft.com/fwlink/p/?linkid=116898&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportMediaExtenderStreaming

WFP-based products must support media streaming scenarios based on extender technologies.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Extender technology is built into home entertainment devices such as TVs, DVD players, and cool, quiet components that allow you to keep your PC where it makes sense and use it as a "hub" to provide your digital entertainment to TVs throughout your house. These devices are called extender devices.

For example: With the new Extenders for Windows Media Center, you can stream the digital media you have on your Windows Media Center PC in as many as five rooms in your house. Home-users may access the live and recorded TV, music, movies, videos, sports, Internet TV and other online content on Windows® PCs through wired or wireless home networks. Windows Media Center Extenders use network ports to communicate with Windows PCs. The following exceptions tabled below may be useful in meeting this requirement:

Media Center Extender SPECIFIC			
Binary	**Port**	**Direction**	**Scope**
svchost.exe (ssdpsrv)	UDP 1900	Inbound	Local Subnet
svchost.exe (termservice)	TCP 3390	Inbound	Local Subnet
svchost.exe (QWave)	TCP 2177	Outbound, Inbound	Local Subnet
svchost.exe (QWave)	UDP 2177	Outbound, Inbound	Local Subnet
System	TCP 10244	Outbound, Inbound	Local Subnet
ehshell.exe	TCP 554	Outbound, Inbound	Local Subnet

ehshell.exe	UDP 5004, 5005	Outbound, Inbound	Local Subnet
ehshell.exe	TCP 8554-8558	Outbound, Inbound	Local Subnet
ehshell.exe	UDP 50004-50013	Outbound, Inbound	Local Subnet
ehshell.exe	UDP 7777-7781	Outbound, Inbound	Local Subnet
mcrmgr.exe	random	Outbound	Internet
mc2prov.exe	random	Outbound	Internet
Svchost.exe (mcs2svc)	random	Outbound	Local Subnet
Media Center Binaries/Ports			
ehrecvr.exe	random	Outbound	Internet
ehrec.exe	random	Outbound	Internet
ehexthost.exe	random	Outbound, Inbound	Internet
mcupdate.exe	random	Outbound	Internet
Digital Cable Receiver Device (OCUR)			
ehprivjob.exe	UDP 5001-5006	Inbound	Local Subnet
svchost.exe	UDP 1900	Outbound, Inbound	Local Subnet
System	TCP 2869	Outbound, Inbound	Local Subnet

ehprivjob.exe	TCP 554	Outbound	Local Subnet
ehprivjob.exe	UDP 5757-5772	Outbound	Local Subnet

The word 'support' refers to the host firewall's capability to make internet streaming & media sharing for media player network sharing services, work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportMobileBroadBand

WFP-based products must allow mobile broadband devices that are compliant with Windows mobile broadband driver model to function correctly.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products must allow mobile broadband devices that are compliant with the Windows mobile broadband driver model to function correctly.

This is to ensure that host firewall functionality does not block the mobile broadband connectivity and the firewall functionality works with MB devices.

Windows provides native support for mobile broadband (MB) data cards & embedded modules to work with Windows. The MB devices need to implement their driver as per Windows mobile broadband driver model. The MB driver model defines how the devices should be exposed to Windows and network packet format in which MB devices should exchange data between network and system.

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportPeerNameResolution

WFP-based products must support Peer Name Resolution Protocol and the Peer-to-Peer Grouping Protocol.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Host firewalls support the Peer Name Resolution Protocol (PNRP) and the Peer-to-Peer Grouping Protocol, which are required by some Peer-to-Peer applications. The Peer Name Resolution Protocol provides secure, serverless name resolution, and the Peer-to-Peer Grouping Protocol provides secure, reliable multi-party communication. The following guidelines may be useful in meeting this requirement:

- Host firewalls support native IPv6 (NETWORK-0244) as well as Teredo (NETWORK-0248) and IPv6 packets to IPv4 protocol 41 (^to4) (NETWORK-0249).
- Host firewalls can allow for the system to send outbound, and receive inbound, UDP packets over port 3540.
- Host firewalls can allow for the system to send outbound, and receive inbound, TCP packets over port 3587.

The word 'support' refers to the host firewall's capability to allow successful DHCP exchanges, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

Please refer to the following documents for more information, these documents:

- HomeGroup firewall requirements: http://technet.microsoft.com/en-us/appcompat/default.aspx
- Network location dialog: http://technet.microsoft.com/en-us/appcompat/default.aspx
- PNRP: http://technet.microsoft.com/en-us/appcompat/default.aspx

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportRemoteAssistance

WFP-based products must support Remote Assistance scenarios.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The Remote Assistance scenario is used by a helper to connect to a computer and to show the user a solution to the problem. The following guidelines may help meet this requirement:

Host firewalls allow the computer to be reached by native IPv4, native IPv6, Teredo, and 6to4 (pass the corresponding tests) and also allow traffic from the Remote Assistance application within Windows® (msra.exe) through the firewall.

The word 'support' refers to the host firewall's capability to make Remote Assistance work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

For information on how Remote Assistance works in general, please see the article below:

http://go.microsoft.com/fwlink/p/?linkid=116842&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportRemoteDesktop

WFP-based products must support remote desktop.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Remote Desktop Connection is a technology that allows you to connect to a remote computer in a different location. Remote desktop is a key Windows® scenario that would be relevant for consumers with multiple PCs at home trying to access content that exists on one PC, from another PC.

The following guideline may help meet this requirement:

Host firewalls allow inbound TCP packets over Destination Port 3389 to support this scenario. This scenario will need to work with native IPv4, native IPv6, 6to4, and Teredo packets.

The word 'support' refers to the host firewall's capability to make remote desktop work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

For more information on remote desktop, please see the article below:

http://go.microsoft.com/fwlink/p/?linkid=116841&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportTeredo

WFP-based products must support Teredo.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

Teredo may be used as a connectivity mechanism to support certain Windows scenarios such as remote assistance, instant messaging and others. Hence, preserving Teredo connectivity is critical to supporting Windows consumer scenarios.

For this requirement, the following must be met:

- Host firewalls allow DNS resolution of teredo.ipv6.microsoft.com.
- To allow client to Teredo server communication, host firewalls must allow for the system to send outbound UDP/IPv4 packets to UDP port 3544.
- To allow Teredo connectivity, host firewalls must allow inbound and outbound UDP/IPv4 traffic over the Teredo client system ports. These ports can be obtained using the FWPMSystemPortsGet notification to determine the system port numbers used for communication using the Teredo interface.
- Host firewalls support ICMP error messages & discovery functions (NETWORK-0250 logo requirement).
- Host firewalls allow UPnP framework packets over UDP port 1900, and UPnP frameworkpackets over TCP port 2869.

The word 'support' refers to the host firewall's capability to make Teredo work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

Please refer to the following article below for further information on Teredo:

http://go.microsoft.com/fwlink/p/?linkid=116836&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.SupportVirtualPrivateNetworking

WFP-based products must support VPN scenarios in Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following protocols and ports must be allowed:

- IP protocol 50: Allow ESP traffic
- IP protocol 51: Allow AH traffic
- UDP Port 500 / 4500: Allow ISAKMP traffic
- TCP / UDP Port 88: Allow Kerberos traffic

This ensures that firewalls support IPsec scenarios, such as IPsec VPN, which are used on client PCs to connect securely over the internet.

In addition, host firewalls should allow successful IPsec communication over both IPv4 and IPv6. Host firewalls should also allow UDP packets over port 1701, and TCP packets over port 443 to support this scenario. It is also recommended that host firewalls allow TCP packets specific over port 1723. IP protocol 47 based packets should also be allowed by the host firewall.

The word 'support' refers to the host firewall's capability to make the VPN scenarios work, if the application/user/network needs it. The host firewall must also have properly configured objects such as filters, etc to support the required functionality, even though the functionality may not be enabled by default in the UI.

Design Notes:

Please refer to the following article for further information:
http://go.microsoft.com/fwlink/p/?linkid=116843&clcid=0x409

Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.InteropWithOtherExtensions

WFP must not block traffic from another vSwitch extension (WFP or LWF) by default, and should only do so when following specific user/admin intention or protecting the system against a specific threat.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP must not block traffic from another vSwitch extension (WFP or LWF) by default, and should only do so when following specific user/admin intention or protecting the system against a specific threat.

Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.NoEgressModification

WFP-based products that operate in the vSwitch must not modify packets on the Egress path of the vSwitch.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products that operate in the vSwitch must not modify packets on the Egress path of the vSwitch.

Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.SupportLiveMigration

WFP-based products that operate in the vSwitch must present a minimal MOF for Live Migration.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products that operate in the vSwitch must present a minimal MOF for Live Migration. In the MOF, it must declare itself Logo compliant for Live Migration and allow itself to be migrated or not block migration by default. The total time for migrations for Live Migration cannot be longer than 2 seconds.

Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.SupportRemoval

WFP-based products that operate in the vSwitch must be allowed to be removed when the admin disabled WFP for the vSwitch instance.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products that operate in the vSwitch must be allowed to be removed when the admin disabled WFP for the vSwitch instance.

Filter.Driver.WindowsFilteringPlatform.Scenario.vSwitch.SupportReordering

WFP-based products that operate in the vSwitch must respond to WFP vmSwitch reorder events.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WFP-based products that operate in the vmSwitch must respond to WFP vmSwitch reorder events.

Send comments about this topic to Microsoft

Components and peripherals

In this section

- Device.Audio.APO
- Device.Audio.Base
- Device.Audio.HardwareAudioProcessing
- Device.Audio.HDAudio
- Device.Audio.USB
- Device.BusController.Bluetooth.Base
- Device.BusController.Bluetooth.NonUSB
- Device.BusController.Bluetooth.USB
- Device.BusController.I2C
- Device.BusController.NFC.NearFieldProximity
- Device.BusController.NFC.RadioManagement
- Device.BusController.NFC.SecureElement.UICC
- Device.BusController.NFC.SmartCard
- Device.BusController.SdioController
- Device.BusController.UART
- Device.BusController.UsbController
- Device.Cluster
- Device.Connectivity.BluetoothDevices
- Device.Connectivity.Network.VerticalPairing
- Device.Connectivity.PciConnected
- Device.Connectivity.Server
- Device.Connectivity.UsbDevices
- Device.Connectivity.UsbHub
- Device.Connectivity.WSD

- Device.DevFund.CDA
- Device.Devfund.DeviceGuard
- Device.DevFund.DriverFramework.KMDF
- Device.DevFund.DriverFramework.UMDF
- Device.DevFund.Firmware
- Device.DevFund.INF
- Device.DevFund.Memory
- Device.DevFund.Reliability
- Device.DevFund.Reliability.3rdParty
- Device.DevFund.Reliability.Interrupts
- Device.DevFund.ReliabilityDisk
- Device.DevFund.Security
- Device.DevFund.Server
- Device.DevFund.Server.PCI
- Device.DevFund.Server.StaticTools
- Device.Display.Monitor
- Device.Graphics.AdapterBase
- Device.Graphics.AdapterRender
- Device.Graphics.AdapterRender.D3D101Core
- Device.Graphics.AdapterRender.D3D101WDDM11
- Device.Graphics.AdapterRender.D3D101WDDM12
- Device.Graphics.AdapterRender.D3D10ComputeShader
- Device.Graphics.AdapterRender.D3D10Core
- Device.Graphics.AdapterRender.D3D10D3D11LogicOps
- Device.Graphics.AdapterRender.D3D10Multisampling4X
- Device.Graphics.AdapterRender.D3D10Multisampling8X
- Device.Graphics.AdapterRender.D3D10WDDM11
- Device.Graphics.AdapterRender.D3D10WDDM12
- Device.Graphics.AdapterRender.D3D111Core
- Device.Graphics.AdapterRender.D3D11ASTC
- Device.Graphics.AdapterRender.D3D11ConservativeRasterization

- Device.Graphics.AdapterRender.D3D11Core
- Device.Graphics.AdapterRender.D3D11DoublePrecisionShader
- Device.Graphics.AdapterRender.D3D11DriverCommandLists
- Device.Graphics.AdapterRender.D3D11DriverConcurrentObjectCreation
- Device.Graphics.AdapterRender.D3D11Level9WDDM12
- Device.Graphics.AdapterRender.D3D11Level9WDDM13
- Device.Graphics.AdapterRender.D3D11PartialPrecision
- Device.Graphics.AdapterRender.D3D11RasterizerOrderedViews
- Device.Graphics.AdapterRender.D3D11StencilReference
- Device.Graphics.AdapterRender.D3D11TypedUAVLoad
- Device.Graphics.AdapterRender.D3D11WDDM12
- Device.Graphics.AdapterRender.D3D11WDDM12DoublePrecisionShader
- Device.Graphics.AdapterRender.D3D11WDDM13
- Device.Graphics.AdapterRender.D3D11WDDM20
- Device.Graphics.AdapterRender.D3D12ASTC
- Device.Graphics.AdapterRender.D3D12ConservativeRasterization
- Device.Graphics.AdapterRender.D3D12Core
- Device.Graphics.AdapterRender.D3D12Multiadapter
- Device.Graphics.AdapterRender.D3D12RasterizerOrderedViews
- Device.Graphics.AdapterRender.D3D12StencilReference
- Device.Graphics.AdapterRender.D3D12TypedUAVLoad
- Device.Graphics.AdapterRender.D312VolumeTiledResources
- Device.Graphics.WDDM
- Device.Graphics.WDDM.Display
- Device.Graphics.WDDM.Display.HDMIorDPDCNs
- Device.Graphics.WDDM.DisplayRender
- Device.Graphics.WDDM.Render
- Device.Graphics.WDDM11
- Device.Graphics.WDDM11.Display
- Device.Graphics.WDDM11.DisplayRender
- Device.Graphics.WDDM11.DisplayRender.D3D9Overlay

- Device.Graphics.WDDM11.Render
- Device.Graphics.WDDM11.Render.DXVAHD
- Device.Graphics.WDDM12
- Device.Graphics.WDDM12.Display
- Device.Graphics.WDDM12.DisplayOnly
- Device.Graphics.WDDM12.DisplayRender
- Device.Graphics.WDDM12.DisplayRender.ProcessingStereoscopicVideoContent
- Device.Graphics.WDDM12.DisplayRender.RuntimePowerMgmt
- Device.Graphics.WDDM12.Render
- Device.Graphics.WDDM12.RenderOnly
- Device.Graphics.WDDM12.StandbyHibernateFlags
- Device.Graphics.WDDM13
- Device.Graphics.WDDM13.DisplayRender
- Device.Graphics.WDDM13.DisplayRender.CoolingInterface
- Device.Graphics.WDDM13.DisplayRender.WirelessDisplay
- Device.Graphics.WDDM13.EnhancedPowerManagement
- Device.Graphics.WDDM13.Render
- Device.Graphics.WDDM20
- Device.Graphics.WDDM20.Core
- Device.Graphics.WDDM20.Display.VirtualModeSupport
- Device.Graphics.WDDM20.DisplayRender
- Device.Imaging.Printer.Base
- Device.Imaging.Printer.Mobile
- Device.Imaging.Printer.Mobile.WSD20
- Device.Imaging.Printer.OXPS
- Device.Imaging.Printer.USB
- Device.Imaging.Printer.WSD
- Device.Imaging.Printer.XPS
- Device.Imaging.Scanner.Base
- Device.Imaging.Scanner.WSD
- Device.Input.Digitizer.Base

- Device.Input.Digitizer.Pen
- Device.Input.Digitizer.PrecisionTouchpad
- Device.Input.Digitizer.Touch
- Device.Input.FingerPrintReader
- Device.Input.HID
- Device.Input.Keyboard
- Device.Input.Location
- Device.Input.PointDraw
- Device.Input.SmartCardMiniDriver
- Device.Input.SmartCardReader
- Device.Network.DevFund
- Device.Network.LAN
- Device.Network.LAN.Base
- Device.Network.LAN.ChecksumOffload
- Device.Network.LAN.CS
- Device.Network.LAN.DCB
- Device.Network.LAN.GRE
- Device.Network.LAN.IPsec
- Device.Network.LAN.KRDMA
- Device.Network.LAN.LargeSendOffload
- Device.Network.LAN.MTUSize
- Device.Network.LAN.PM
- Device.Network.LAN.RSC
- Device.Network.LAN.RSS
- Device.Network.LAN.SRIOV
- Device.Network.LAN.SRIOV.VF
- Device.Network.LAN.TCPChimney
- Device.Network.LAN.VMQ
- Device.Network.LAN.VXLAN
- Device.Network.MobileBroadband.CDMA
- Device.Network.MobileBroadband.FirmwareUpdater

- Device.Network.MobileBroadband.GSM
- Device.Network.Switch.Manageability
- Device.Network.WLAN
- Device.Network.WLAN.SupportConnectionToAP
- Device.Network.WLAN.SupportDot11W
- Device.Network.WLAN.SupportFIPS
- Device.Network.WLAN.SupportHostedNetwork
- Device.Network.WLAN.SupportHotspot2Dot0
- Device.Network.WLAN.SupportMACAddressRandomization
- Device.Network.WLAN.SupportWakeFromLowPower
- Device.Network.WLAN.SupportWiFiDirect
- Device.Network.WLAN.SupportWiFiDirectServices
- Device.Portable.Core
- Device.Portable.DigitalCamera
- Device.Portable.DigitalVideoCamera
- Device.Portable.MediaPlayer
- Device.Portable.MobilePhone
- Device.Storage.Controller
- Device.Storage.Controller.Ata
- Device.Storage.Controller.Boot
- Device.Storage.Controller.Fc
- Device.Storage.Controller.Fc.NPIV
- Device.Storage.Controller.Fcoe
- Device.Storage.Controller.Flush
- Device.Storage.Controller.Iscsi
- Device.Storage.Controller.Iscsi.iSCSIBootComponent
- Device.Storage.Controller.Optical
- Device.Storage.Controller.PassThroughSupport
- Device.Storage.Controller.Raid
- Device.Storage.Controller.Raid.ContinuousAvailability
- Device.Storage.Controller.Sas

- Device.Storage.Controller.Sata
- Device.Storage.Controller.SD
- Device.Storage.ControllerDrive.NVMe
- Device.Storage.Enclosure
- Device.Storage.Hd
- Device.Storage.Hd.1394
- Device.Storage.Hd.Alua
- Device.Storage.Hd.Ata
- Device.Storage.Hd.AtaProtocol
- Device.Storage.Hd.DataVerification
- Device.Storage.Hd.Ehdd
- Device.Storage.Hd.EMMC
- Device.Storage.Hd.EnhancedStorage
- Device.Storage.Hd.FibreChannel
- Device.Storage.Hd.Flush
- Device.Storage.Hd.Iscsi
- Device.Storage.Hd.Mpio
- Device.Storage.Hd.MultipleAccess
- Device.Storage.Hd.MultipleAccess.PersistentReservation
- Device.Storage.Hd.OffloadedDataTransfer
- Device.Storage.Hd.PersistentReservation
- Device.Storage.Hd.PortAssociation
- Device.Storage.Hd.RaidArray
- Device.Storage.Hd.ReadZeroOnTrimUnmap
- Device.Storage.Hd.RemovableMedia
- Device.Storage.Hd.Sas
- Device.Storage.Hd.Sata
- Device.Storage.Hd.Sata.HybridInformation
- Device.Storage.Hd.Scsi
- Device.Storage.Hd.Scsi.ReliabilityCounters
- Device.Storage.Hd.ScsiProtocol

Send comments about this topic to Microsoft

Device.Audio.APO

This APO must match all APO tests.

In this topic:

Device.Audio.APO.MicArrayRawData

System effect in the capture path provides RAW data from microphone array when requested by the client.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a microphone array processing algorithm is provided in a Windows system effect audio processing object (APO) instantiated in a stream effect (SFX) insert point in capture path, it must provide all the individual audio streams as channels from the array when a client asks for a format with number of channels equal to the number of microphone elements in the array. This allows the APO to provide hardware compensation processing and microphone array processing to the client that takes advantage of the entire APO, but allows clients that rely on the microphone array processing that resides higher up in the audio subsystem to take advantage of hardware compensation in the APO but not the array processing in it.

It is highly recommened for onboard fixed-position microphone array (multiple combined elements) on a system to follow “Speech Platform Device Recommendations Specification” for an optimized experience.

Send comments about this topic to Microsoft

Device.Audio.Base

This device must match all base tests.

In this topic:

Device.Audio.Base.AudioProcessing

Audio devices must support proper audio processing discovery and control.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Drivers

At a minimum, a driver must support a raw mode or a default mode pin. On the pins provided to the system the hardware must supply a post-mix volume if it supports a) mixing or b) audio offload. Also, the hardware must supply mute on the render side, if it supports a) mixing or b) audio offload or c) compressed.

Endpoint effects must be scenario neutral. The endpoint effects must work in all scenarios. For effects that may harm a scenario such as real time communications or only be beneficial to one scenario, the effect must be placed into the mode effects that are specific to that scenario. The only allowed Endpoint effects (EFX) and Raw Mode Effects (MFX in raw mode) are speaker compensation and speaker protection.

In addition drivers that support the RAW mode must support the following depending on your driver structure:

- A driver that supports mixing without offload support (supports multiple concurrent modes but does not support offload) shall include a KSNODETYPE_SUM node and no KSNODETYPE_AUDIO_ENGINE node. The node shall have a single input connection coming from the software pin factory, and represents the point where multiple instances of the pin are mixed.
- The KSNODETYPE_AUDIO_ENGINE or KSNODETYPE_SUM node shall be in the path between the software pin factories and the endpoint bridge pin. The node shall be in the same filter as the software pin factories.
- The node shall support KSPROPERTY_AUDIOSIGNALPROCESSING_MODES.
    - For Port Class drivers, the miniport shall support IMiniportAudioSignalProcessing. The port shall add KSPROPERTY_AUDIOSIGNALPROCESSING_MODES to the appropriate pin.
- A driver that supports mixing (with offload and/or multiple modes) shall support a loopback pin.
- Loopback pins shall be in new pin category KSPINCATEGORY_AUDIOLOOPBACK. The topology shall have a path from the software pin factory to the loopback pin factory.
- A driver that does not support mixing shall support KSPROPERTY_AUDIOSIGNALPROCESSING_MODES on the endpoint's software pin factory.
    - For Port Class drivers, the miniport shall support IMiniportAudioSignalProcessing. The port shall add KSPROPERTY_AUDIOSIGNALPROCESSING_MODES to the appropriate pin.
- The software pin factory data ranges shall include KSATTRIBUTE_AUDIO_SIGNALPROCESSINGMODE. The attribute shall not be marked KSATTRIBUTE_REQUIRED.

- The pin creation code shall check for KSDATAFORMAT_ATTRIBUTES in the data format and process the KSATTRIBUTE_AUDIOSIGNALPROCESSINGMODE if present.

For Port Class drivers, the driver's implementation of NewStream on IMiniportWaveRT, IMiniportWavePci, or IMiniportWaveCyclic shall support KSDATAFORMAT_ATTRIBUTES.APOs

- If the driver's APOs support DEFAULT then the offload pin shall support DEFAULT.
- The APOs shall support all modes that are supported by the offload pin.
- APOs shall support all the modes supported by the host pin.

Discovery

Driver must expose all audio effect via the FXStreamCLSID, FXModeCLSID, and FXEndpointCLSID APOs (or proxy APOs). The APOs must send an accurate list of effects that are enabled to the system when queried. Drivers must support APO change notifications and only notify the system when an APO change has occurred.

Loopback

The loopback stream should represent the stream coming out of the speaker. Drivers with hardware processing must provide the system an accurate loopback stream.

Non-offload drivers that support mixing must support DRMRIGHTS on all pin instances. If any stream in the graph on a given pin instance requires loopback constriction, then the audio system asserts DRMRIGHTS.CopyProtect on that pin.

Device.Audio.Base.DRM

Audio device must implement DRM support as defined in the Windows Driver Kit.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Audio devices must comply with Windows trusted audio paths for digital rights management (DRM). Hardware that complies with Windows DRM supports DRM level 1300. The audio drivers must not call the DrmForwardContentToFileObject function.

The DRM requirement does not apply if the underlying device is a Bluetooth audio device.

Design Notes:

See the"Digital Rights Management" topic in the Windows Driver Kit.

Device.Audio.Base.Endpoints

Audio subsystem properly reflects current system configuration

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Requirements Device.Audio.Base.JackDetection, Device.Audio.Base.NoUncontrollableStreamRouting and Device.Audio.Base.ExposedAudioEndpointsAreFunctional have been merged with this one.

All Audio Devices

Any audio device that is exposed by any of the audio device enumeration APIs must reflect the current state of all of its endpoints appropriately at all times when the system is powered on. This includes the case, when the system is in connected standby. Built-in speakers and microphones must work while the system is operational.

If an endpoint is exposed by any of the device enumeration APIs and its state reflects being connected and capable of streaming, it must be functional (capable of capture/render). Exposed audio end points need to continue to function even during system state changes such as:

- While the power source changes from external to battery or vice versa
- While the GPU switches from a to b

Devices that use Windows v10.0 Mobile

The audio driver must not perform hidden stream redirection, routing, switching, splitting, mixing, muxing to other exposed or hidden logical audio devices, applications or other entities but it must ensure that the audio stream from the audio system endpoint for a particular logical device is only directed to that particular logical device that the application is streaming to, as set by the Windows user in the Windows Sound control panel.

Headphone connectors, speaker connectors, HDMI connectors, DisplayPort connectors, devices connected through a docking station, wireless and network connected audio devices

The driver needs to properly express the connection state of the corresponding devices.

If the connector is not plugged, or the wireless or network device is not in a connected usable state then the driver must either:

- Set the KSJACK_DESCRIPTION.IsConnected member to FALSE, or
- Disable the KS filter interface
    - For AvStream drivers: Declare a connector is unplugged by implementing the following properties and events:
        - KSPROPSETID_Jack
            - KSPROPERTY_JACK_DESCRIPTION

        - KSPROPERTY_JACK_DESCRIPTION2
    - KSEVENTSETID_PinCapsChange
        - KSEVENT_PINCAPS_JACKINFOCHANGE
- For Port Class drivers: Unregister the Port Class subdevice by calling PcUnregisterSubdevice

Similarly, when the connector is plugged, or the wireless or network device is connected an in a usable state, then the driver must either:

- Set the KSJACK_DESCRIPTION.IsConnected member to TRUE, or
- Enable the KS filter interface:
    - For AvStream drivers: Declare a connector is plugged by implementing the following properties and events:
        - KSPROPSETID_Jack
            - KSPROPERTY_JACK_DESCRIPTION
            - KSPROPERTY_JACK_DESCRIPTION2
        - KSEVENTSETID_PinCapsChange
            - KSEVENT_PINCAPS_JACKINFOCHANGE
    - For Port Class drivers: Registers the Port Class subdevice by calling PcRegisterSubdevice

Whenever the driver changes the KSJACK_DESCRIPTION.IsConnected member, the driver must generate the event KSEVENTSETID_PinCapsChange / KSEVENT_PINCAPS_JACKINFOCHANGE.

The above behavior ensures Windows routes audio to the connector or device only when it is truly available for streaming.

For connectors and wireless or network devices, mechanism (a) is preferred. This makes the device visible in the system UI (e.g. the Windows Sound control panel), even though it might not be immediately usable, and it ensures that the device is displayed with correct status.

For devices that are in a detachable docking station, mechanism (b) is recommended.

For connectors that are on a detachable docking station, mechanism (b) is recommended to reflect the attachment of the dock. Mechanism (a) is recommended to reflect whether the connector is plugged.

Design notes: See the Microsoft UAA HD Audio Pin Configuration Programming Guidelines white paper for additional clarifications on the specified jack connectors that require jack detection.

http://www.microsoft.com/whdc/device/audio/PinConfig.mspx

Device.Audio.Base.HardwareArchitecture

Audio subsystems must use a technology compatible with Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Requirement **Device.Audio.UAACompliance.UAA** has been replaced with this one.

Integrated audio devices

An integrated audio device is one that supports an internal component or a port that's used exclusively for media content. Here are some examples of integrated audio devices:

- Speakers
- Microphones and microphone arrays
- Analog audio jacks (headphone jack, line out, line in, microphone jack)
- S/PDIF
- Digital outputs like HDMI and DisplayPort

For these devices, any audio hardware architecture can be used provided at least one of the following is true:

- The device provides basic functionality for all of its endpoints when used with any of the audio class drivers packaged with Windows.
- A driver is available through Windows Update that will enable basic functionality for all the device's endpoints.

Externally connected audio devices

An externally connected audio device is one that isn't integrated to the system and has a connection that isn't specific to audio or media. Here are some examples of external audio devices:

- USB audio
- Bluetooth audio

For these devices, any audio hardware architecture can be used, but we strongly recommended that these devices conform to standard specifications and provide basic functionality with a Windows audio class driver. On certain Windows devices that don't allow the installation of a third-party drivers, the only way for an external audio device to function is if it's compatible with a class driver.

All audio devices

If the PnP ID of an audio device is compatible with any of the audio class drivers packaged with Windows, the device must provide basic functionality for all of its endpoints when using that driver.

A device provides basic functionality when it meets all the Windows Hardware Certification audio device requirements.

For more info

Audio Device Technologies: http://msdn.microsoft.com/en-us/library/windows/hardware/gg454527.aspx

Device.Audio.Base.PowerManagement

Audio device must comply with related power management specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All Audio Devices and Drivers

Audio devices must comply with Audio Device Class Power Management Reference Specification, which provides definitions of the device power states (D0–D3) for these devices. The specification also covers the device functionality expected in each power state and the possible wake-up event definitions for the class. The device and driver must implement support for power state D3. Support for other device power management states is optional.

Bluetooth Audio Devices

Bluetooth audio devices must complete an HCIDisconnect before powering down.

The HCIDisconnect is required to allow for timely notification to the system that the device is no longer available. This is used to reroute audio to an alternate audio sink seamlessly when the Bluetooth audio device is powered off.

Reference(s)

ACPI Specification: http://www.acpi.info/

Bluetooth Specifications: https://www.bluetooth.org/en-us/specification/adopted-specifications

MSDN: http://msdn.microsoft.com/en-us/library/windows/hardware/gg463220.aspx

Device.Audio.Base.SamplePositionAccuracy

Audio driver reports render sample position with defined accuracy for stream synchronization.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The **Device.Audio.Base.SamplingAccuracy** requirement has been merged with this one.

For all audio endpoints, IAudioClock::GetPosition shall report timestamps with:

- |bias| <= 1ms
- |skew| <= 1%
- Jitter <= 1ms

This requirement applies to both render and capture for all formats, modes, and pins (host, offload, loopback).

Device.Audio.Base.StreamingFormats

Audio subsystems must use formats supported by Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following requirements have been merged with this one:

- Device.Audio.Base.TwoDMAEnginesAndConnections
- Device.Audio.Base.BasicDataFormats
- Device.Audio.Base.ChannelMasks

All Audio Devices

This requirement applies to both input (capture) and output (render) devices.

An audio device must expose support for at least one PCM (Pulse Code Modulation) encoded format compatible with Windows.

At least one of the following bit depths and containers must be used:

- 8 bit (unsigned)
- 16 bit
- 20 bit in a 24 bit container
- 24 bit
- 24 bit in a 32 bit container

- 32 bit

The samples must be either integer or IEEE 754 float.

Any sampling rate will work with the windows audio pipeline; however, it is suggested that 48 kHz and 44.1 kHz be supported for optimal power performance in media scenarios.

If the device supports both input and output capabilities, the audio device must support independent selection of formats and support concurrent streaming at arbitrarily selected formats.

Channel configurations must be at least one of the following:

- (mono)
- (stereo)
- 2.1
- 3.1
- 4.0
- 5.0
- 5.1
- 7.1

If the audio device supports multichannel audio formats, the audio device driver must deal with channel masks consistent with the content and the current selected speaker configuration.

Digital Devices with External Connections

For devices where there exists a source and sink relationship (such as HDMI and Bluetooth), a source device must support and expose all required sink formats. For example, HDMI requires a sink support 48 kHz or 44.1 kHz; therefore, an HDMI source must support (at a minimum) both 48 kHz and 44.1 kHz so that any HDMI sink device connected will function properly. It is suggested that sources support all possible sink formats for the best user experience.

Hardware Acceleration (Offloading)

Devices that leverage a DSP for audio offloading must support a superset of formats that are supported by the host (system) pin on the same device. This ensures that offloading is used whenever possible and avoids unnecessary fall back to the host pin.

Reference(s)

MSDN: http://msdn.microsoft.com/en-us/library/windows/hardware/ff536189(v=vs.85).aspx

Device.Audio.Base.VolumeControl

Audio driver volume controls are linear and have adequate resolution.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Requirement Device.Audio.Base.VolumeGranularity has been merged with this one.

Signal response (as measured by electrical or digital signal level) changes in linearity with the volume control within 3% tolerance. For example: a volume slider change of 10dB should result in a measured volume change between 9.7 dB and 10.3 dB + or – 0.3dB.

Topology volume nodes must have a resolution equal to or better than 1.5 dB and implement driver support for volume level as defined in the Windows Driver Kit.

See the Windows Driver Kit, "KSPROPERTY_AUDIO_VOLUMELEVEL" for more details.

Send comments about this topic to Microsoft

Device.Audio.HardwareAudioProcessing

HardwareAudioProcessing

In this topic:

Device.Audio.HardwareAudioProcessing.AudioHardwareOffloading

Hardware that supports offloaded audio render processing must meet this requirement.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The requirement Device.Audio.HardwareAudioProcessing.IMiniport.xml has been merged with this one.

If a hardware solution supports offloaded audio render processing, the driver must expose a KS filter and a single KSNODETYPE_AUDIO_ENGINE node with appropriate pin factories connected.

If a hardware solution supports the offloading of audio render processing, mixing, or decoding, the driver must expose a KS filter. For each rendering path through that filter that supports hardware offloading the driver must expose a single KSNODETYPE_AUDIO_ENGINE node, connecting directly to only the following pin factories:

- Two KS sink pin factories
- a single KS source pin factory for reference stream support

If a driver exposes a KSNODETYPE_AUDIO_ENGINE node, the driver and hardware must support base-level functionality.

If a driver exposes a KSNODETYPE_AUDIO_ENGINE node, the driver and hardware must support the following capabilities:

- Audio mixer with at least 3 simultaneous inputs (2 offload and 1 host process)
- Volume and mute capabilities both pre- and post-mixing
- Reference stream (support for sending the audio stream post-mix back to the Windows audio stack)
- The reference stream provided should be the final output to the audio device, or, if encoding is taking place, just prior to encoding.

If hardware supports metering reporting (support for querying per-stream peak values, both pre & post-mix)

- For stream metering (pre-mixing), metering levels should be reported after the SFX and before volume control
- For endpoint metering (post-mixing), metering levels should be reported:
  - Before volume control and EFX, when the EFX is an encoder
  - After the EFX and before volume control, when the EFX is not an encoder

If a driver exposes a KSNODETYPE_AUDIO_ENGINE node, the driver must expose certain pin factories.

If a driver exposes a KSNODETYPE_AUDIO_ENGINE node, the driver must expose the following pin factories:

- Host process pin factory
  - Must support at least one instance
- Offload pin factory
  - Must support at least two instances
- Loopback pin factory
  - Must support at least a single instance

In addition, the following must be met:

- Loopback pins must:
  - Have a “Possible Global Instances” of at least 1
  - Support at least 1 instance regardless of what else is going on in the system
- To enable scenarios like cross-fade, offload-capable endpoints must support 1 loopback pin instance + 1 host pin instance + each of the following in isolation, assuming no other offload endpoints are being used at the time:
  - Any of supported PCM format + Any of supported PCM format (the same, or different)
- The loopback pin must support
  - The HW mix format
  - The device format (which can be publically queried from the endpoint property store)

If a hardware solution supports offloaded audio render processing, the same functionality provided in hardware (e.g., processing, effects, etc.) must be available on the host pin as well.

In order to provide a consistent user experience and prevent confusion when a user enables or configures functionality that exists in only hardware or only software, the capabilities provided must be equal in both hardware and software.

Other Considerations

Offloaded audio devices must accept and properly react to end of segment (EOS) communication from the operating system.

If a hardware solution supports offloaded audio render processing, the driver must implement IMiniportAudioEngineNode and IMiniportStreamAudioEngineNode2

IMiniportAudioEngineNode contains a list of methods related to the offload KS properties targeting the audio engine node via KS filter handle. A miniport driver's WaveRT miniport class needs to inherit not only from IMiniportWaveRT interface, it also needs to inherit IMiniportAudioEngineNode interface and implement all the defined methods.

Send comments about this topic to Microsoft

Device.Audio.HDAudio

This audio device uses the HD audio driver.

In this topic:

Device.Audio.HDAudio.HDAudioSpecCompliance

HD Audio codec for audio must comply with the Intel High Definition Audio specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following requirements have been merged with this one:

- Device.Audio.HDAudio.HDAudioCodecAdditionalReqs
- Device.Audio.HDAudio.HDMIDCN
- Device.Audio.HDAudio.INFHasDeviceID
- Device.Audio.AudioController.HDControllerCompliance

A **High Definition Audio codec** must comply with the following specifications:

- Intel High Definition Audio Specification and DCNs
- Plug and Play Guidelines for High Definition Audio Devices

Additionally, the code must implement the following features, which are not necessarily required by the Intel High Definition Audio Specification:

Speaker compensation is the only valid scenario for audio signal processing of an audio stream by a codec, and then it is valid only if the speakers are hardwired to the pin complex that contains the processing node (such as integrated laptop speakers). This requirement does not apply to the decryption of protected audio streams.

- When all of an HDAudio codec's widgets are configured in the benign processing state, the codec performs no nonlinear or time-variant processing on the audio streams that pass through it.
- An HDAudio codec must be accessible only through the HDAudio bus controller. The codec must not expose registers or other hardware mechanisms that are accessible through either memory or I/O address space. This requirement does not encompass HDMI or DisplayPort. For HDMI or DisplayPort, please refer to the HD audio HDMI DCN.
- Modem and audio functionality must not be combined. Although the same piece of silicon can house both modem and audio devices, the functions must be separate devices and must not share any software or hardware resources (such as ADCs or DACs).

- When the HD Audio link is in a running state (HD Audio controller is in D0), HD Audio codecs must respond to commands even when powered down in all required device power-management states. In effect, the digital section of the codec must remain powered.
- Codecs must respond to a verb even if addressed at a nonexistent widget or if the verb itself is invalid.
- Function group nodes must have node IDs in the range 0 to 127. This restriction does not apply to node IDs for widget nodes.
- The default data in the HD Audio codec pin configuration registers must not misrepresent the hardware capabilities, and the Configuration Default Registers must not be null (all zeros).
- A function group in an HDAudio codec must expose a nonzero subsystem ID. The BIOS overwrites the subsystem ID if necessary. If the BIOS cannot program the subsystem ID or if it does so incorrectly, the hardware must supply a default, vendor-specific subsystem ID.
- Each HD Audio codec port connects to one and only one audio source, destination, or jack. For compatibility with the class driver, do not double-up on input or output ports in ways that cannot be exposed to the class driver through the information in the pin configuration registers. Designs that use GPIOs under control of third-party function drivers must default to an appropriate hardware configuration when the class driver is loaded.
- HD Audio Codec Driver Must Not Leave Function Group in D3Cold State Upon Unload. By the exit of the IRP handler for IRP_MJ_PNP/IRP_MN_REMOVE_DEVICE, an HD Audio Codec driver must have:
    - Remembered or discovered the current power state of the function group
    - If that current function group power state was D3 Cold, the driver must have changed it to a different power state. The function group power state upon exit is required to be D3.

HD Audio controllers must comply with the following requirements:

- Intel High Definition Audio Controller specification
- Be updated to comply with future specification revisions
- Comply with the latest HD Audio specification ECRs in accordance with policies around new hardware requirements.

- HD Audio hardware that complies with HD Audio specification version 1.0 must set the correct version number in the appropriate registers. The VMAJ and VMIN registers must specify a major version number of 01h and a minor version number of 00h.

Send comments about this topic to Microsoft

Device.Audio.USB

This audio device uses the USB audio driver.

In this topic:

Device.Audio.USB.HIDControls

USB audio device uses USB HID audio controls to keep the operating system informed of user interactions with the device.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Requirement **Device.Audio.USB.HIDCommunications** has been merged with this one.

USB audio devices must use USB HID specification-compliant HID to control basic functions. If volume adjustment controls are implemented on the USB audio device, it must declare itself as a consumer control device (usage 0x01), as defined in Consumer Page (page 0x0C) in the USB Usage Tables for HID Power Devices, Release 1.1, and in Windows support for HID-based audio controls.

Communication devices that implement a USB HID interface must be compliant with the USB Device Class Definition for Human Interface Devices (HID), Version 1.1, and USB Usage Tables for HID Power Devices, Version 1.12.

Devices may not use Reserved usages from any Standard Usage Page.

See "HID Audio Controls and Windows" at http://go.microsoft.com/fwlink/?LinkId=40491 and the Windows Driver Kit, "HID and Windows" for more design information.

Device.Audio.USB.USB

USB audio device must follow UAA USB audio design guidelines.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Requirements Device.Audio.Base.ProperUSBDescriptors and Device.Audio.USB.MicArray have been merged with this one.

Description: A USB audio-based audio device in a stand-alone external form factor, or in an AVR or in other permutations complies with the Device.Audio.Base.HardwareArchitecture.

Special attention should be made to the following:

- USB audio device must properly set descriptor to indicate the purpose of device according to the USB spec http://www.usb.org/developers/devclass_docs/termt10.pdf.
- An externally connected USB based microphone array device must comply with the USB Device Class Definition for Audio Devices 2.0, and must be implemented according to the guidelines in "Microphone Array Support in Windows Vista." The device must report itself and its capabilities according to the design guidelines in the Microsoft USB Audio Microphone Array Design Guidelines.

Reference:

Universal Serial Bus Device Class Definition for Audio Devices 2.0 at http://www.usb.org/developers/docs/devclass_docs/

Send comments about this topic to Microsoft

Device.BusController.Bluetooth.Base

In this topic:

- Device.BusController.Bluetooth.Base.Scatternet
- Device.BusController.Bluetooth.Base.SimultaneousBrEdrAndLeTraffic
- Device.BusController.Bluetooth.Base.SpecificInformationParameters
- Device.BusController.Bluetooth.Base.SupportsBluetooth21AndEdr

Device.BusController.Bluetooth.Base.4LeSpecification

Bluetooth controllers must support the Bluetooth 4.0 specification requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

These requirements are "If Implemented" for Client systems and apply only if a Client system supports Bluetooth.

The Bluetooth controller must comply with the Basic Rate (BR) and Low Energy (LE) Combined Core Configuration Controller Parts and Host/Controller Interface (HCI) Core Configuration requirements outlined in the Compliance Bluetooth Version 4.0 specifications.

The Bluetooth radio HW shall be qualified as a "Controller Subsystem" and may additionally be qualified as a "Component" through the Bluetooth Special Interest Group.

Device.BusController.Bluetooth.Base.LeStateCombinations

Bluetooth controllers must support a minimum set of LE state combinations.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth controller must allow the spec LE state combinations (as allowed in section [Vol 6] Part B, Section 1.1.1 of the Bluetooth version 4.0 spec): Only the following states are not required to be supported:

- 0x0000000000800000 Active Scanning State and Initiating State combination supported.
- 0x0000000004000000 Passive Scanning state and Slave Role combination supported.
- 0x0000000008000000 Active Scanning state and Slave Role combination supported.

Device.BusController.Bluetooth.Base.LeWhiteList

Bluetooth controllers must support a minimum LE white list size of 25 entries.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth controller must support a minimum of 25 entries in its white list for remote Low Energy (LE) devices.

Device.BusController.Bluetooth.Base.MicrosoftBluetoothStack

Bluetooth controllers must be tested using Microsoft's Bluetooth stack.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth controllers must be tested with Microsoft's Bluetooth stack when submitting for hardware certification.

Device.BusController.Bluetooth.Base.HCIExtensions [If Implemented]

Microsoft defined HCI extensions support for hardware offload of advertisement and RSSI monitoring.

Applies to	Windows 10 x64 Windows 10 x86

Description

Radios that support the Microsoft defined Bluetooth HCI extensions must comply with the specification and pass the related HLK tests. Specification details can be found on MSDN:

- https://msdn.microsoft.com/en-us/library/windows/hardware/dn917903(v=vs.85).aspx

Device.BusController.Bluetooth.Base.NoBluetoothLEFilterDriver

Bluetooth LE filter drivers are not allowed to load on BTHLEENUM.SYS.

Applies to	Windows 10 x64 Windows 10 x86

Description

To ensure a uniform experience across Windows Store Apps using the Bluetooth LE (GATT) WinRT API, filter drivers shall not be loaded on BTHLEENUM.SYS.

Device.BusController.Bluetooth.Base.OnOffStateControllableViaSoftware

Bluetooth controllers' On/Off state must be controllable via software.

Applies to	Windows 10 x64 Windows 10 x86

Description

When turning the radio "off", Bluetooth controllers shall be powered down to its lowest supported power state and no transmission/reception shall take place. Windows will terminate Bluetooth activity by unloading the inbox protocol drivers and their children, submitting the HCI_Reset command to the controller, and then setting the controller to the D3 logical power state, allowing bus drivers to power down the radio as appropriate. The radio can be completely powered off if a bus-supported method is available to turn the radio back on. No additional vendor software control components will be supported.

On turning the radio back on, the Windows Bluetooth stack shall resume the device to D0, allowing bus drivers to restart the device. The Windows Bluetooth stack shall then reinitialize the Bluetooth components of the controller.

Bluetooth Radio Management in Windows 8.1 shall only be enabled for internal Bluetooth 4.0 controllers.

Device.BusController.Bluetooth.Base.Scatternet

Bluetooth host controller must support Bluetooth scatternet.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth host controller must support at least two concurrent piconets (also known as a scatternet). The host controller must also be able to allow the host to join a device that is requesting a connection to the existing piconet when the local radio is the master of that piconet. This requirement is described in the Specification of the Bluetooth System, Version 2.1 + Enhanced Data Rate (EDR) (Baseband Specification), Section 8.6.6. Design Notes: The scatternet support should follow the enhanced scatternet support errata that are defined by the Bluetooth Special Interest Group (SIG).

Device.BusController.Bluetooth.Base.SimultaneousBrEdrAndLeTraffic

Bluetooth controllers must support simultaneous BR/EDR and LE traffic.

Applies to	Windows 10 x64 Windows 10 x86

Description

Bluetooth controllers must allow the simultaneous use of both Basic Rate (BR)/Enhanced Data Rate (EDR) and Low Energy (LE) radios.

Device.BusController.Bluetooth.Base.SpecificInformationParameters

Bluetooth host controller must implement specific Informational parameters to provide accurate information about the host controller's capabilities.

Applies to	Windows 10 x64 Windows 10 x86

Description

The manufacturer fixes the informational parameters, which provide valuable information about the Bluetooth device and the capabilities of the host controller. Bluetooth host controllers must implement the HCl_Read_Local_Version_Information command and HCI_Read_Local_Supported_Features command as described in the Specification of the Bluetooth System, Version 2.1 + Enhanced Data Rate (EDR), Part E, Section 7.4. Required support includes the mechanism for reporting the supported version and features.

Device.BusController.Bluetooth.Base.SupportsBluetooth21AndEdr

Bluetooth controllers must support the Bluetooth 2.1+EDR specification requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth host controller must comply with the requirements that are outlined in the Specification of the Bluetooth System Version 2.1 + Enhanced Data Rate (EDR).

Send comments about this topic to Microsoft

Device.BusController.Bluetooth.NonUSB

Bluetooth Controller - NonUSB connected radios

In this topic:

Device.BusController.Bluetooth.NonUSB.Performance

Non-USB Bluetooth controllers must achieve at least a throughput of 700 kbps.

Applies to	Windows 10 x64 Windows 10 x86

Description

Non-USB Bluetooth controllers must achieve at least a throughput of 700 kbps at the RFCOMM layer.

Device.BusController.Bluetooth.NonUSB.ScoSupport

Non-USB connected Bluetooth controllers must use the sideband channel for SCO.

Applies to	Windows 10 x64 Windows 10 x86

Description

In order to ensure a high quality audio experience, all non-USB connected Bluetooth controllers must use a sideband channel for SCO (e.g. SCO over an I2S/PCM interface).

Send comments about this topic to Microsoft

Device.BusController.Bluetooth.USB

Bluetooth Controller - USB connected radios

In this topic:

Device.BusController.Bluetooth.USB.ScoDataTransportLayer

Bluetooth host controllers must support the SCO data transport layer as specified in the Bluetooth 2.1+EDR specifications.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth host controller must comply with the Synchronous Connection Oriented (SCO)-USB requirements that are outlined in the specification of the Bluetooth System, Version 2.1 + Enhanced Data Rate (EDR), Part A, Section 3.5.

Send comments about this topic to Microsoft

Device.BusController.I2C

These requirements apply only to I2C controller silicon vendors. System manufacturers may optionally run these tests, but may need hardware customization.

In this topic:

Device.BusController.I2C.CancellationOfIO

I2C controller and controller drivers must support the cancellation of I/O requests.

Applies to	Windows 10 x64 Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following:

- Driver implements SPB request cancelation logic for read/write/sequence I/O.

Device.BusController.I2C.ClockStretching

I2C controller and controller drivers must support peripheral clock stretching.

Applies to	Windows 10 x64 Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following:

- Controller can sustain peripheral holding clock for at least 2 seconds during read, write, and sequence I/O.

Device.BusController.I2C.HCKTestability

Systems with I2C controllers must expose correct ACPI table information and I2C pin-outs to enable HCK testability.

Applies to	Windows 10 x64 Windows 10 x86

Description

The objective of this requirement is to enable the controller to be testable by the HCK framework.

Details:

- Controller under test must provide I2C external connectivity pin-out (SCL, SDA, and GND).
- Update ACPI to correctly describe HCK test peripheral drivers and its connection to I2C controller under test.
- Other peripheral devices on the same I2C controller under test must be disabled when running HCK tests.

Device.BusController.I2C.IdlePowerManagement

I2C controller and controller drivers must support Idle Power Management.

Applies to	Windows 10 x64

	Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following:

- Controller should go to the D3 state after it is idle for more than 1 second when the screen is on.
- Controller should go to the D3 state after idle for more than 100ms when the screen is off.
- Controller takes less than 75 ms (50+ 25 to account for the timer granularity of 15ms) to resume from the D3 state to the D0 state.

Device.BusController.I2C.LockUnlockIOCTL

I2C controller and controller drivers must support the Lock/Unlock IOCTL.

Applies to	Windows 10 x64 Windows 10 x86

Description

If the Stop condition is supported, the I2C controller and associated controller driver must conform to the SPB framework and support the following:

- Supports an arbitrary number of read/write operations inside Lock/Unlock pair.
- Generate the Start condition for the first I/O in the lock/unlock sequence, the Restart condition for subsequent I/O, and the Stop condition when Unlock is called.

Device.BusController.I2C.NACK

I2C controller and controller drivers must support peripheral NACK.

Applies to	Windows 10 x64 Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following:

- Controller can detect address NACK bus condition and return STATUS_NO_SUCH_DEVICE for request.
- Controller can detect device NACK during a write operation, complete the request with STATUS_SUCCESS, and information bytes is set to a number of bytes that is less than what was intended to be written.
- Controller can detect device NACK during a write operation of a sequence IOCTL, complete the request with STATUS_SUCCESS, and information bytes is set to number of bytes that is less than what was intended to be written.

Device.BusController.I2C.SPBRead

I2C controller and controller drivers must support SPB Read operations correctly.

Applies to	Windows 10 x64 Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following when reading data from an I2C peripheral:

- Must support reading from standard (100Kbps), fast (400kbps) and fast plus (1 Mbps) peripheral targets. High Speed (3.4 MHz) is optional, but must pass all HCK requirements for I2C if implemented in the I2C controller and controller driver.
- Must support read size from 1 to 4096 bytes (4 KBytes).
- Sizes larger than 4 KBytes must succeed or fail with STATUS_NOT_SUPPORTED.
- SPB read is mapped into Start, Read Data, NACK, and Stop I2C conditions.
- Fail any unsupported data size read request with STATUS_INVALID_PARAMETER and not cause any bus activities.

Device.BusController.I2C.SPBSequenceIOCTL

I2C controller and controller drivers must support SPB Sequence IOCTL correctly.

Applies to	Windows 10 x64 Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following:

- Supports any arbitrary I/O sequences: write-read, read-write, write-write, read-read and complex combined such as write-read-read-write-write
- SPB sequence IOCTL is mapped into Start, I/O sequence 1, Restart….I/O sequence N, Stop I2C conditions.
- Controller needs to examine the sequence and determine if it is supported or fail with STATUS_INVALID_PARAMETER before causing any bus activities.
- Support any valid parameters (e.g. DelayInUs) and memory format (SIMPLE, MDL, Buffer list etc.) as defined by SPB.

Device.BusController.I2C.SPBWrite

I2C controller and controller drivers must support SPB Write operations correctly.

Applies to	Windows 10 x64 Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following when writing to an I2C peripheral:

- Must support writing to standard (100Kbps), fast (400kbps) and fast plus (1 Mbps) peripheral targets. High Speed (3.4 MHz) is optional, but must pass all HCK requirements for I2C if implemented in the I2C controller and controller driver.
- Must support write size from 1 to 4096 bytes (4 KBytes).
- Sizes larger than 4 KBytes must succeed or fail with STATUS_NOT_SUPPORTED.
- SPB write is mapped into Start, Write Data, and Stop I2C conditions.
- Fail any unsupported data size write request with STATUS_INVALID_PARAMETER and not cause any bus activities.

Device.BusController.I2C.Stress

I2C controller and controller driver must operate correctly and recovers from bus hangs or faults under prolonged stress conditions.

Applies to	Windows 10 x64 Windows 10 x86

Description

The I2C controller and associated controller driver must conform to the SPB framework and support the following:

- Supports bus recovery when peripheral is hung (watchdog mechanism).
- Sustain multiple targets stress for more than 1 hour.

Send comments about this topic to Microsoft

Device.BusController.NFC.NearFieldProximity

Any technology that implements the GUID_DEVINTERFACE_NFP device driver interface specified in the NFP Device Driver Requirements is defined as an NFP provider and must meet all the Near Field Proximity DDI implementation requirements laid out within the specification.

In this topic:

Device.BusController.NFC.NearFieldProximity.Attribute [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must support maximum message size no smaller than 10 KB and transmission rate no smaller than 16KB per second.

Device.BusController.NFC.NearFieldProximity.Event [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must enable clients to receive DeviceArrived and DeviceDeparted events upon arrival or departure of a proximate device.

Device.BusController.NFC.NearFieldProximity.NDEF [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must support the NDEF protocol as defined by the NFC Forum in the NDEF specification v1.0.

Device.BusController.NFC.NearFieldProximity.PeerToPeer [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must support transmitting and receiving data via the NFC Forum defined Logical Link Control Protocol (LLCP) v1.1 and Simple NDEF Exchange Protocol (SNEP) v1.0.

Device.BusController.NFC.NearFieldProximity.Publish [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must support publishing and transmitting messages.

Device.BusController.NFC.NearFieldProximity.Reliability [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must reliably read and write tags.

Device.BusController.NFC.NearFieldProximity.Subscribe [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must be able to receive and deliver messages to subscribed clients.

Device.BusController.NFC.NearFieldProximity.TagOperation [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFP provider must be able to read and write tags as well as perform operations such as setting tags to read only.

Send comments about this topic to Microsoft

Device.BusController.NFC.RadioManagement

The Radio Management DDI allows callers to the NFC device driver to set the power state of the proximity radio of the NFC device. An NFC device driver must implement the DDIs defined in the Radio Management DDI document.

In this topic:

Device.BusController.NFC.RadioManagement.Base [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The NFC device driver must set the power states of the proximity and secure element radios of the NFC device.

Send comments about this topic to Microsoft

Device.BusController.NFC.SecureElement.UICC

The Secure Element DDI allows callers to the NFC device driver to enumerate, communicate with, and configure the secure elements accessible from the device.

Any technology that implements the GUID_DEVINTERFACE_NFCSE device driver interface specified in the Secure Element DDI document is defined as an NFC Secure Element provider and must meet all the Secure Element DDI implementation requirements laid out within the specification.

In this topic:

Device.BusController.NFC.SecureElement.UICC.Emulation [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

An NFC Secure Element provider that supports UICC based card emulation must be able to communicate with and configure the UICC and must grant exclusive access to the client to manage card emulation mode..

Device.BusController.NFC.SecureElement.UICC.Enumeration [If Implemented]

Applies to	Windows 10 x64

	Windows 10 x86

Description

An NFC Secure Element provider that supports UICC based card emulation must be able to enumerate all secure elements that are accessible from the device.

Device.BusController.NFC.SecureElement.UICC.Event [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

An NFC Secure Element provider that supports UICC based card emulation must support client subscriptions for events and must be able to raise events to indicate occurrences such as transactions with an external reader.

Device.BusController.NFC.SecureElement.UICC.Reliability [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

An NFC Secure Element provider that supports UICC based card emulation must be able to reliably turn card emulation mode on / off.

Send comments about this topic to Microsoft

Device.BusController.NFC.SmartCard

The Smart Card DDI allows callers to the NFC device driver to perform low-level smart card operations on NFC contactless smart cards. This includes listening on card arrival/departure notifications, reading metadata of the smart card like ATR, UID and Historical Bytes information, performing read/write operations on the specific NFC card using APDUs as well as the support for translating APDUs to low-level primitive commands for some non-ISO14443-4 compliant cards.

Any technology that implements the GUID_DEVINTERFACE_SMARTCARD_READER device driver interface specified in the Smart Card DDI document is defined as a Smart Card Reader provider and must meet all the Smart Card DDI implementation requirements laid out within the specification.

In this topic:

Device.BusController.NFC.SecureElement.SmartCard.Attribute [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The Smart Card Reader provider must be able to get and set smart card attributes.

Device.BusController.NFC.SecureElement.SmartCard.DataExchange [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The Smart Card Reader provider must be able to communicate with supported contactless smart cards.

Device.BusController.NFC.SecureElement.SmartCard.State [If Implemented]

Applies to	Windows 10 x64 Windows 10 x86

Description

The Smart Card Reader provider must be able to get and set the contactless smart card’s state.

Send comments about this topic to Microsoft

Device.BusController.SdioController

In this topic:

Device.BusController.SdioController.ComplyWithIndustrySpec

SDIO controller must comply with the industry standard.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Secure Digital I/O (SDIO) host controllers must comply with PCI 2.3 or later requirements for that interface. For PCI configuration registers and interface information, see the SD Host Controller Specification, Version 1.0, Appendix A.

Device.BusController.SdioController.WdfKmdfDriver

SDIO controller driver must be a WDF KMDF implementation.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The SDIO controller driver must be written using the Windows Driver Framework (WDF) Kernel Mode Driver Framework for the driver's implementation.

Send comments about this topic to Microsoft

Device.BusController.UART

The requirements apply only to silicon vendors. UART controller drivers are recommended to use SerCXV2.

In this topic:

Device.BusController.UART.Cancellation

UART controller and controller drivers must support the cancellation of Read and Write requests.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the following:

- Controller implements necessary logic to support I/O cancellation

Device.BusController.UART.DMA

UART controller and controller drivers require DMA support for appropriate DMA transactions.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the following:

- Peripheral driver can issue read and write request to the controller max at 5 K data size.

Device.BusController.UART.FlowControl

UART controller and controller drivers must support the setting of the flow control to on and off.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the following:

- Driver implements support for IOCTL_SERIAL_GET_HANDFLOW and IOCTL_SERIAL_SET_HANDFLOW IOCTLs and flow control settings

Device.BusController.UART.FlushFIFO

UART controller and controller drivers must support Flush FIFOs.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the ability to flush FIFO queues.

Device.BusController.UART.HCKTestability

Systems with UART controllers must expose correct ACPI table information and UART pin-outs to enable HCK testability.

Applies to	Windows 10 x64 Windows 10 x86

Description

The objective of this requirement is to enable the UART controller to be testable by the HCK framework.

Details:

- Controller under test must provide UART external connectivity pin-out (Rx,Tx, RTS, CTS, and GND).

- Describe HCK UART test peripheral driver and its connection to UART controller under test in device's firmware.

Device.BusController.UART.IdlePowerManagement

UART controller and controller drivers must support Idle Power Management.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the following:

- Controller transitions to the Dx state when there is no pending I/O in the controller for 200 ms.

Device.BusController.UART.Performance

UART controller and controller driver has a measured baud rate that matches the expected value.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework that the measured baud rate matches the expected value.

Device.BusController.UART.ReadWrite

UART controller and controller drivers must support read/write Unicode(8 bits) data.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the following when reading data from an UART peripheral:

- Support IOCTL_SERIAL_SET_LINE_CONTROL and IOCTL_SERIAL_GET_LINE_CONTROL and be able to transfer data according to the data length settings (8 bits).

Device.BusController.UART.Stress

UART controller and controller driver operates correctly (and recovers appropriately from bus errors) under prolonged stress conditions.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the following:

- Sustain stress test passes for at least 1 hour.

Device.BusController.UART.SupportedBaudRates

UART controller and controller drivers must support basic baud rate 115200 and faster speed for higher bandwidth communications.

Applies to	Windows 10 x64 Windows 10 x86

Description

The UART controller and associated controller driver must conform to the Serial framework and support the following:

- Driver supports IOCTL_SERIAL_SET_BAUD_RATE and IOCTL_SERIAL_GET_BAUD_RATE IOCTL.
- Driver should fail baud rate setting for non-supported baud rate and is able perform I/O using the baud rate set.

Send comments about this topic to Microsoft

Device.BusController.UsbController

In this topic:

- Device.BusController.UsbController.PreserveDeviceStateAfterDisableEnable
- Device.BusController.UsbController.UsbifCertification
- Device.BusController.UsbController.TestedUsingMicrosoftUsbStack
- Device.BusController.UsbController.XhciAc64Bit
- Device.BusController.UsbController.XhciAddInCardsMapPortsConsistently
- Device.BusController.UsbController.XhciAddInCardsReportInternalDevices
- Device.BusController.UsbController.XhciSupportDebuggingOnAllExposedPorts
- Device.BusController.UsbController.XhciSupportMsiMsixInterrupts
- Device.BusController.UsbController.XhciSupportsMinimum31Streams
- Device.BusController.UsbController.XhciSupportsRuntimePowerManagement
- Device.BusController.UsbController.XhciVersionCompliant

Device.BusController.UsbController.ImplementAtLeastOneXhciSpcStructForUSB2

xHCI controllers must implement at least one xHCI Supported Protocol Capability structure for USB 2.0.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Extensible Host Controller Interface (xHCI) controllers must implement at least one xHCI supported protocol capability structure for USB 2.0 as described in section 7.2 of the xHCI specification.

This affects backward compatibility with USB 2.0.

Device.BusController.UsbController.MaintainDeviceStateOnResumeS1andS3

USB host controller must maintain device state on resume from S1 or S3.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

For the host controller to maintain the device state, the USB host controller must not issue a USB bus reset on a system sleep state transition from S3 to S0 or from S1 to S0.

USB host controllers that are integrated or embedded into the south bridge chipset must decouple the USB bus reset from the PCI bus reset to reduce resume time. Resume operations from S1 or S3 must not generate USB bus resets. A USB bus reset is a reset signal that is sent over the USB bus to USB devices that are connected to the USB host controller.

Systems that have a USB keyboard attached are allowed to perform USB bus resets to unlock the system by using a password when the system resumes from S3.

For security purposes, the BIOS in a mobile system is allowed to issue a USB bus reset if the system is attached to a docking station that has a hard disk drive (HDD) that is password-locked on first resume.

A reset of the HDD password is allowed whether or not the mobile system is docked. The following scenarios are allowed:

- Undocked systems with a password-enabled HDD
- Docked systems with a password-enabled HDD
- Addition or removal of an HDD

If the docking station does not have a native HDD or the docking station does not have a password, the BIOS must not issue a USB bus reset.

It is acceptable to allow the controller to lose power in S3 when the system is on battery power.

Design Notes:

See the Enhanced Host Controller Interface Specification for Universal Serial Bus, Revision 1.0, Appendix A.

This requirement does not apply to systems that support Connected Standby.

Device.BusController.UsbController.MustResumeWithoutForcedReset

All USB host controllers must work properly upon resume from sleep, hibernation, or restart without a forced reset.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All USB host controllers work properly upon resume from sleep, hibernation, or restart without a forced reset of the USB host controller.

Design Notes:

A reset of the entire USB host controller results in significantly increased time that it takes for all USB devices to become available after system resume since there could be only one device at address 0 at a time, this enumeration has to be serialized for all USB devices on the bus. We have also seen that resetting the host controller can lead to an illegal SE1 signal state on some host controllers, which in turn can cause some USB devices to hang or drop off the bus. Moreover, devices cannot maintain any private state across sleep resume as that state will be lost on reset.

Device.BusController.UsbController.PreserveDeviceStateAfterDisableEnable

USB controller must preserve device states after a disable and re-enable.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a USB controller is disabled and then re-enabled, all devices that were attached to the controller before the USB controller was disabled are required to be present after the USB controller is re-enabled.

Device.BusController.UsbController.UsbifCertification

USB host controller is USB IF certified.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB host controllers must pass USB Implementers Forum (IF) testing.

For details, see the following link:

http://msdn.microsoft.com/en-us/windows/hardware/gg463175.aspx

Note: Since USB-IF is currently not certifying controllers for Windows on ARM systems, the Windows on ARM controllers are exempt from needing to get full USB-IF certification. Instead, the WoA controllers are expected to pass all Windows Hardware Certification tests which include eventing, loop back, and registers tests that get run as part of USB-IF certification.

Device.BusController.UsbController.TestedUsingMicrosoftUsbStack

xHCI controllers must be tested with Microsoft's xHCI Stack installed.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Extensible Host Controller Interface (xHCI) controllers must be tested with Microsoft's xHCI stack installed and enabled on a Windows system. Support for all USB transfer types (ISoch, Interrupt, and Bulk) will be checked to ensure basic compatibility.

Device.BusController.UsbController.XhciAc64Bit

xHCI controllers must set the AC64 bit in the HCCPARAMS register to 1.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

xHCI controllers must set the AC64 bit in the HCCPARAMS register to 1 as described in Section 5.3.6 of the xHCI specification.

Therefore, the controller must support:

- 64-bit addressing, described in section 5.3.6
- 64-bit register access, described in section 5.1

Design notes:

Checking for AC64 to be set is a simple register check in the compliance driver.

To test 64-bit addressing, we will need to require the HLK user's client system to have at least 6 GB of RAM. The test will use MmAllocateContiguousMemorySpecifyCache to get physical memory above 4 GB. It will validate in some way that the controller can access this memory area.

The test will try writing one or more registers using a 64-bit register access and reading back using 64-bit register access to confirm that registers are updated correctly. An example of a reasonable register to test is: "**Event Ring Segment Table Base Address Register (ERSTBA)**" (section 5.3.2.3.2).

If AC64 is not set, there is nothing to test.

Device.BusController.UsbController.XhciAddInCardsMapPortsConsistently

xHCI add-in cards must map USB 3.0 and USB 2.0 ports consistently.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

Consistent USB 2.0 and USB 3.0 port mapping is required to help the operating system to effectively manage the ports.

Note: This requirement only applies to add-in cards because port mapping for integrated xHCI controllers should be performed via Advanced Configuration and Power Interface (ACPI). For more information, see the SYSFUND 226 requirement.

For Extensible Host Controller Interface (xHCI) add-in cards (where "add-in card" is defined as a card that is not integrated onto the motherboard), the complexity of this requirement varies significantly depending on whether the add-in card contains any internal (integrated or embedded) hubs.

If there are no internal hubs, then the port numbering must correlate as given in xHCI v1.x specification. That is, the first USB 3.0 port must be connected to the same connector as the first 2.0 port, the second with the second, and so on. For example, if the USB 2.0 ports are numbered 1 and 2, and the USB 3.0 ports are numbered 3 and 4, ports 1 and 3 must map to the same connector, and ports 2 and 4 must map to the same connector. For more information, see the xHCI v1.x Specification, sections 4.24.2.1 and 4.24.2.2. If the host does not have any internal hubs, then the remaining text of this requirement can be ignored.

However, if there are internal hubs (either integrated or embedded), then the requirement is more involved. Note that strictly speaking, XHCI specification does not allow such hubs for add-in cards because the port mapping information cannot be communicated to the software via ACPI. But through this requirement, we are allowing such hubs and defining the required port mapping. However, this mechanism has some limitations and it does not allow arbitrary configurations that are allowed for integrated controllers when described by ACPI.

For add-in cards, xHCI host controllers may implement "integrated hubs" and/or "embedded hubs" as defined in xHCI specification sections 4.24.2.1 and 4.24.2.2. Embedded hubs need not be limited to being on the system board. However, the following limitations apply:

- Embedded hubs of add-in cards must be USB 3.0 hubs (this limitation is unique to the scenario of this requirement and not part of the xHCI specification).
- An add-in card may have at most 1 integrated hub.
- If an add-in card has an integrated hub, it must have only 1 USB2 protocol port on the root hub. This port is the port connected to the integrated hub.
- An add-in xHCI card that implements an integrated hub must set the Integrated Hub Implemented (IHI) bit in the USB 2.0 xHCI Supported Protocol Capability structure to '1' for the root hub port connected to an integrated hub (refer to section 7.2.2.1.3.2 of the xHCI specification).
- All integrated or embedded hubs must be marked non-removable in their parent ports.

The implementation of integrated hubs determines the External Ports of the controller. External Ports are a concept defined in section 4.24.2 of the xHCI specification to order ports, so that they can be mapped to connectors. In all cases, let there be n USB2 protocol External Ports numbered 1 to n, and m USB3 protocol External Ports numbered n+1 to n+m.

External Port numbers are assigned to meet the following properties (not defined in the xHCI specification). Note that integrated hubs must be USB 2.0 hubs.

- If the xHCI implements an integrated hub, then n, the number of USB2 protocol External Ports, equals the number of downstream facing ports on the integrated hub.
- Otherwise, n equals the number of downstream facing USB2 protocol ports on the root hub.
- m, the number of USB3 protocol External Ports, equals the number of downstream facing USB3 protocol ports on the root hub.
- Assign External Port numbers such that External Ports 1 through n are USB2 protocol ports and External Ports n+1 through n+m are USB3 protocol external ports, and the ordering ports within each protocol is preserved.

If embedded hub(s) are not present: The USB2 protocol External Ports and USB3 protocol External Ports must be mapped to connectors using the "default" mapping described in section 4.24.2.2 of the xHCI specification under the heading "When an Embedded hub is not implemented".

If embedded hub(s) are present: The embedded hubs must be USB 3.0 hubs. First, determine the connector mapping as it would be without any embedded hubs, using the "default" mapping from section 4.24.2.2 of the xHCI specification. For each embedded hub, both upstream ports must be connected to the same connector. The embedded hubs' downstream ports map to new connectors in the same way as the ports of a non-embedded USB 3.0 hub.

Non-exposed connectors: Devices embedded in the host controller must be marked non-removable in their parent ports. If, according to the connector mapping above, a non-removable peripheral device's connector is shared with a second port, the second port must not be connected or connectable to any device. On the other hand, any connector whose port(s) are all marked as removable is considered to be an exposed connector, i.e. it must be physically connectable.

Note that if there is no ACPI information, a root hub cannot have both an embedded USB2 device and an integrated USB2 hub; instead, the embedded device must be attached to the integrated hub.

Device.BusController.UsbController.XhciAddInCardsReportInternalDevices

xHCI controller add-in cards must correctly report internally attached devices.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Extensible Host Controller Interface (xHCI) controllers must indicate internally attached devices by setting the device removable (DR) bit in the PORTSC register to 1 for every port that has an internally attached device. This applies to controllers that do not have ACPI information. For more information, see section 5.4.8 of the xHCI Specification.

- This requirement will prevent the operating system from flagging non-removable devices as removable.
- Add-in cards are defined as host controllers that are not integrated onto the motherboard.

Design Notes:

Note: This requirement only applies to add-in cards because port mapping for integrated xHCI controllers should be performed via Advanced Configuration and Power Interface (ACPI).

Device.BusController.UsbController.XhciSupportDebuggingOnAllExposedPorts

xHCI controllers must support USB debugging on all exposed ports.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Extensible Host Controller Interface (xHCI) host controllers are debug-capable on all ports. Ports that have embedded non-removable devices attached do not need to report debug capability.

- USB debugging is defined in section 7.6 of the xHCI specification.
- This requirement does not apply to add-in card host controllers.

Device.BusController.UsbController.XhciSupportMsiMsixInterrupts

xHCI controllers must support MSI and/or MSI-X interrupts.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Extensible Host Controller Interface (xHCI) controllers support Message Signaled Interrupts (MSI) and MSI-X interrupts as defined in section 6.8 of the PCI Local Bus Specification, revision 3.0 and section 5.2.6 of the xHCI Specification.

Device.BusController.UsbController.XhciSupportsMinimum31Streams

xHCI controllers must support at least 31 primary streams per endpoint.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Refer to the eXtensible Host Controller Interface specification, section 4.12.2.

This requirement is for the MaxPSASize in the HCCPARAMS to be set to 4 at the minimum to enable ultimate data transfer rate with UAS devices.

Storage devices based on the USB Attached SCSI Protocol (UASP) will utilize streams to achieve faster data transfer rates. To enable the best experience with these devices, every xHCI controller will need to support at least 31 primary streams.

Device.BusController.UsbController.XhciSupportsRuntimePowerManagement

USB xHCI host controllers must support runtime power management including, if implemented, runtime wake.

Applies to	Windows 10 x64 Windows 10 x86

Description

All USB xHCI host controllers must support runtime power management, as required by the eXtensible Host Controller Interface specification, version 1.0, Section 4.15.

Runtime is defined as the system working state (S0), including the Connected Standby sub-state of S0 if the controller is tested on a system that supports Connected Standby.

Power management of the host controller encompasses software-initiated idle power down (controller low power state such as D3), software-initiated power up, and, optionally, hardware-initiated wake signaling.

If the xHCI controller is reported to support runtime wake signaling, it must be able to wake itself successfully upon any of the following events:

A) Any port detecting device wake signaling

B) Any port detecting connect, disconnect, or overcurrent, when the corresponding PORTSC Wake on Xxx bit is set to '1'.

For more details, see Section 4.15 of the xHCI specification.

To report whether the controller supports runtime wake signaling:

- For add-in controllers, the controller's PCI configuration space must accurately report whether the controller is capable of waking up via PME. Note: reporting that the controller supports waking up via PME implies that the controller can both successfully perform PCI wake at runtime, and successfully wake the system from a system low power state, in accordance with the appropriate PCI specification.

- For integrated controllers, the ACPI _S0W object must report whether the controller is capable of runtime wake signaling.

Device.BusController.UsbController.XhciVersionCompliant

USB 3.0 controllers are XHCI version compliant.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB 3.0 controllers must comply with the Extensible Host Controller Interface (xHCI) Specification version 1.0 and any USB-IF Errata that are released by the USB-IF.

Send comments about this topic to Microsoft

Device.Cluster

In this topic:

Device.Cluster.Core.ClusterInABox

Cluster in a Box

Applies to	Windows Server 2016 Technical Preview x64

Description

A minimum of two or more Windows Server Certified systems grouped together in a shared storage cluster, running the Failover Clustering feature of Windows Storage Server, Windows Server, or Hyper-V Server, as a pre-configured Cluster in a Box product that is self-contained and purchasable as a single SKU, must have:

All components of the clustered server systems, such as devices, associated drivers, and filter drivers that are used in any system in the Cluster in a Box product, or are part of the Network or Storage Connected Devices of the Cluster in a Box product, and that have a defined Product Type must be certified for the version of Windows Server for which the Cluster in a Box product is being tested and submitted for certification.

All components of the clustered server systems, such as devices, associated drivers, and filter drivers that are used in any system in the Cluster in a Box product, but which do not have a defined Product Type must have a signature provided by Microsoft for the version of Windows Server for which the Cluster in a Box product is being tested and submitted.

For all the components above with a defined Product Type, and all those components above without a defined Product Type, all testing must be done using the Hardware Lab Kit [or successor] for the version of Windows Server operating system for which the Cluster in a Box product is being submitted.

All server systems must have all supported features of the included devices, drivers, filters and Network and Storage Connected Devices enabled during server system testing.

All server systems used in the Cluster in a Box product must be certified for the version of Windows Server for which the Cluster in a Box product is being submitted.

All server systems used in the Cluster in a Box testing must have all supported features of the included devices, drivers, filters and Network and Storage Connected Devices enabled and used during testing.

The complete Cluster in a Box product with all components must pass the Cluster in a Box tests included in the Hardware Lab Kit [or successor], in order for the test results submitted to Microsoft to be awarded the certification for a Cluster in a Box product.

Send comments about this topic to Microsoft

Device.Connectivity.BluetoothDevices

Devices that connect to the PC via Bluetooth.

In this topic:

Device.Connectivity.BluetoothDevices.BluetoothDeviceIdProfileVer13

Devices which support Bluetooth must implement the Device ID (DI) profile version 1.3 or Device Information Service (DIS), as applicable.

Applies to	Windows 10 x64 Windows 10 x86

Description

Bluetooth PC peripherals must include the Device ID record as specified in the Device ID profile, version 1.3, for BR/EDR Bluetooth or the Device Information Service (DIS), version 1.1, Bluetooth LE.

Device.Connectivity.BluetoothDevices.BluetoothHidLimitedDiscoverableMode

Bluetooth HID devices must be discoverable only in Limited Discoverable Mode.

Applies to	Windows 10 x64 Windows 10 x86

Description

Bluetooth HID devices must be discoverable only in Limited Discoverable Mode.

Device.Connectivity.BluetoothDevices.ComplementarySubsystemList

Bluetooth wireless technology subsystem end product must list Windows operating system in its complementary subsystem list.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth subsystem end product must list the Windows operating system in the complementary subsystem list as described in Bluetooth Qualification Program Reference Document, Version 2.1, Section 6.1, "Bluetooth Subsystems."

Device.Connectivity.BluetoothDevices.HidInitiatedReconnect

HID devices that support Bluetooth must support HID-initiated re-connect.

Applies to	Windows 10 x64 Windows 10 x86

Description

The HIDReconnectInitiate attribute (defined in Bluetooth HID Profile, 1.0, Section7.11.5, "HIDReconnectInitiate") must be enabled. To automatically reconnect to the host if the connection is dropped, the device must enter the page mode.

Device.Connectivity.BluetoothDevices.KeyboardsSupportPasskeyAuthentication

Bluetooth keyboards that implement Secure Simplified Pairing must support the Passkey authentication method.

Applies to	Windows 10 x64 Windows 10 x86

Description

Keyboards that implement Secure Simplified Pairing must support the Passkey authentication method.

Device.Connectivity.BluetoothDevices.SupportBluetooth21

Devicesthat support Bluetooth must implement the Bluetooth 2.1 requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth devices must comply with the Bluetooth 2.1 + EDR requirements outlined in Bluetooth Version 2.1 + EDR specifications.

Send comments about this topic to Microsoft

Device.Connectivity.Network.VerticalPairing

Root for former Rally technologies

In this topic:

Device.Connectivity.Network.VerticalPairing.WCN

An 802.11 network-enabled device that operates as a station (STA) must implement WCN-NET and meet basic 802.11 requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

An 802.11 network-enabled device that operates as a station (STA) must meet the following requirements:

- The device must implement WCN-NET and comply with the specification.
- The device must implement WCN-NET vertical-pairing extensions and indicate whether it supports a PnP-X transport protocol. If the device supports a PnP-X transport protocol, it must ensure correct universally unique identifier (UUID) alignment.
- If WCN-UFD is implemented, it must comply with the specification.
- If the device has a display that is capable of showing a four-digit or eight-digit number, it must support displaying a dynamic Windows Connect Now (WCN) PIN without user intervention. The PIN must be displayed for a minimum of two minutes after the device receives a Wireless Provisioning Services (WPS) M2D message with the value of "Windows" in the WPS Model Name attribute.
- If the device does not have a display that is capable of showing a four-digit or eight-digit number, it must provide a physical label affixed to the device that includes the eight-digit PIN and clearly labels the PIN value as a PIN (for example, PIN: 12345670).
- The device must be certified by the Wi-Fi Alliance, including Wi-Fi certification, Wi-Fi Protected Access 2 (WPA2) certification, and Wi-Fi Protected Setup certification.

Design Notes:

For implementation details, see the WCN-NET specification at http://go.microsoft.com/fwlink/?LinkId=109371 and the WCN-UFD specification at http://go.microsoft.com/fwlink/?LinkId=109372.

For more information, see the "Installing Wi-Fi Devices Using Rally Vertical Pairing" white paper at http://www.microsoft.com/whdc/connect/rally/WiFiVerticalPair.mspx.

Additional information can be found at http://go.microsoft.com/fwlink/?LinkId=109373 and http://go.microsoft.com/fwlink/?LinkId=109368.

WCN-NET is required. WCN-UFD is optional and is supported in Windows for backward compatibility with devices that are designed to support WCN functionality for Windows XP with Service Pack 2.

A device uses WCN-NET vertical-pairing extensions to indicate that its supports PnP-X. The device must provide a single UUID that is provided in both the WCN-NET exchange and the UPnP/Web Services for Devices (WSD) device file or provide the UPnP/WSD device UUID in the TransportUUID

attribute of the WCN-NET vertical-pairing extension. The UUID that is provided in UPnP or WSD must be in lowercase (decimal digits can also be used).

For WSD implementations, the WSD UUID is provided as the endpoint reference address and must be of the form urn:uuid:. For UPnP implementations, the UPnP UUID is provided as the root device UUID.

Send comments about this topic to Microsoft

Device.Connectivity.PciConnected

In this topic:

Device.Connectivity.PciConnected.64BitPrefetchableBar

PCI-X and PCI Express devices that use prefetchable memory BARs, implement 64-bit prefetchable memory base address registers (BARs)

Applies to	Windows Server 2016 Technical Preview x64

Description

Devices that sit on the PCI-X or PCI Express bus and use prefetchable memory BARs must implement 64-bit prefetchable memory BARs on x64-based systems.

Design Notes:

See "Firmware Allocation of PCI Device Resources in Windows"

http://www.microsoft.com/whdc/system/bus/pci/PCI-rsc.mspx

If the device supports 64-bit prefetchable memory BARs, Windows attempts to assign a region above 4 GB. In a PCI bridge, Windows ignores boot configuration for an entire device path emanating from the bridge in whose scope this method is defined. For the bridge and devices below it to be assigned a region above 4 GB, all devices in the path must support 64-bit prefetchable BARs. If this is not true,

the rebalance code runs and moves all resource assignments below 4 GB, because the goal is to start as many devices as possible

Device.Connectivity.PciConnected.ConfigurationSpaceCorrectlyPopulated

Configuration space for PCI device is correctly populated

Applies to	Windows Server 2016 Technical Preview x64

Description

PCI2.3 describes the configuration space that the system uses to identify and configure each device that is attached to the bus. The configuration space is made up of a header region and a device-dependent region. Each configuration space must have a 64-byte header at offset0. All the device registers that the device circuit uses for initialization, configuration, and catastrophic error handling must fit within the space between byte64 and byte255.

All other registers that the device uses during normal operation must be located in normal I/O or memory space. Unimplemented registers or reads to reserved registers must finish normally and return zero. Writes to reserved registers must finish normally, and the data must be discarded.

All registers that the device requires at interrupt time must be in I/O or memory space.

Device.Connectivity.PciConnected.ExpressCardImplementsSerialNumber

A single ExpressCard module that supports both USB and PCI Express interfaces implements a common serial number

Applies to	Windows Server 2016 Technical Preview x64

Description

An ExpressCard module that supports both USB and PCI Express interfaces on a single module must implement the common serial number as described in the PCI ExpressCard Electromechanical Specification.

This is the only method that Windows will use to determine the relationship of USB and PCI Express on one module.

Device.Connectivity.PciConnected.InterruptDisableBit

PCI and PCI-X devices, that are PCI 2.3 compliant, support the interrupt-disable bit

Applies to	Windows Server 2016 Technical Preview x64

Description

All PCI and PCI-X devices that claim support for PCI Local Bus Specification Revision 2.3 or later, must support the interrupt-disable bit.

Design Notes:

See PCI Local Bus Specification, Revision2.3, Section6.2.2.

Device.Connectivity.PciConnected.MsiOrMsixSupport

PCI device that reports PCI-X capability in the PCI configuration space and that generates interrupts may support MSI or MSI-X but is not required to do so

Applies to	Windows Server 2016 Technical Preview x64

Description

As part of the PCI Conventional Specification 2.2 or later, PCI-X Addendum, Section 3.2, all PCI-X devices that generate interrupts must support message-signaled interrupts.

For MSI implementation, MSI capabilities must be implemented in the configuration space.

For MSI-X implementation, MSI-X capabilities must be implemented in the configuration space.

However, because PCI-X is being replaced by PCI Express and many existing implementations do not support MSI or MSI-X, this requirement is being relaxed. Any device that claims to support MSI or MSI-X must do so or will fail the relevant WDK tests.

Design Notes:

Message Signaled Interrupt for PCI-X device is required by industry standard specification. However, see above.

Device.Connectivity.PciConnected.PciAndPcixDevicesArePciCompliant

PCI and PCI-X devices, at a minimum, are PCI 2.1 compliant

Applies to	Windows Server 2016 Technical Preview x64

Description

All PCI and PCI-X devices must comply with the PCI Local Bus Specification, Revision 2.1 or later. This requirement applies to devices on X86, IA64 and x64 systems.

Design Notes:

See PCI Local Bus Specification, Revision 2.1 or later.

Device.Connectivity.PciConnected.PCIExpress

PCI Express requirement

Applies to	Windows Server 2016 Technical Preview x64

Description

1. Device driver for PCI Express device does not modify VC Enable settings:

The device driver must not modify the "VC Enable" bit (PCI Express Base Specification, Version1.0a, Section7.11.7, VC Resource Control Register: bit 31) for any of the device's extended (non-VC0) virtual channel or channels.

2.PCI Express link active state L1 exit latency does not exceed 64 µS:

A PCI Express link, between root complex and device or bridge, cannot have an active state L1 exit latency of more than 64 microseconds on systems unless the link is associated with a PCI Express cable; that is, a value of 111b cannot be reported in the link capabilities register field 17:15. See PCIe Express Base Specification, Revision 1.0a, Section 7.8.6.

3.PCI Express hot-plug port that supports firmware-controlled hot plug uses the _OSC control method for enable and disable:

All PCI Express hot-plug ports that support firmware-controlled hot-plugs must support the_OSC control method for enabling and disabling firmware-controlled hot-plug as described in the PCI Firmware Specification Version 3.0. See PCI Express Base Specification, Revision 1.1, Section 6.7.4.

4.PCI Express component implements a single device number on its primary interface:

Every PCI Express component (that is, physical device) is restricted to implementing a single device number on its primary interface (upstream port), but it may implement up to eight independent functions within that device number. See PCI Express Base Specification, Revision1.1 (or later), Section 7.3.1.

5. PCI Express device implements support for MSI or MSI-X:

MSI support, which is optional for PCI 2.1, PCI 2.2, and PCI 2.3 devices, is required for PCI Express devices. This capability can be either implemented as MSI or MSI-X. See PCI Express Base Specification, Revision1.0a, Section 6.1.

6. **PCI Express root complex supports the enhanced (memory-mapped) configuration space access mechanism:**

The root complex must support the enhanced configuration space access mechanism as defined in PCI Express Base Specification, Revision 1.1 (or later), Section 7.9.

7.PCI Express device that can interrupt supports the interrupt disable bit:

If an interrupt is implemented, PCI Express devices must support the interrupt disable functionality described in PCI Local Bus Specification, Revision2.3. This bit disables the device or function from asserting INTx. A value of 0 enables the assertion of its INTx signal. A value of 1 disables the assertion of its INTx signal. This bit's state upon reset is 0

Device.Connectivity.PciConnected.SubsystemIdsRequired

Device IDs include PCI subsystem IDs

Applies to	Windows Server 2016 Technical Preview x64

Description

The SID and SVID fields must comply with the SID requirement in PCI Local Bus Specification 2.3 and the implementation details in "PCI Device Subsystem IDs and Windows."

AMR devices and MR devices on the system board are not exempt from the requirement for SID and SVID.

SVID is not required for PCIe to PCI/PCI-X bridges.

Design Notes:

See "PCI Device Subsystem IDs and Windows" at http://go.microsoft.com/fwlink/?LinkId=36804.

Send comments about this topic to Microsoft

Device.Connectivity.Server

In this topic:

Device.Connectivity.Server.ServerOutOfBandManageability

Server Baseboard Management Controller (BMC) devices must support out-of-band management capabilities.

Applies to	Windows Server 2016 Technical Preview x64

Description

BMC devices must support server hardware out-of-band management capability, using IPMI 2.0 through a LAN and/or Serial interfaces, as well as in-band through the KCS system channel via the IPMI Driver.

It is not necessary that the BMC implements the full IPMI 2.0 specification, as only a subset of functionality is required for out-of-band management. The BMC must support the following capabilities:

- The system’s boot source is configurable through the BMC.
    - The BMC must support the following operations being performed on the server system.

      This functionality is implemented using the Set System Boot Options command.
        - The server can be configured to boot from the PXE server the next time it is reset
        - The server can be configured to boot from the hard-disk the next time it is reset
        - The server can be configured to always boot from the PXE server

        - The server can be configured to always boot from the hard-disk
- The system's BMC firmware and BIOS version information is exposed through the BMC.
    - The BMC must expose the version information for the following components:
        - BIOS (through Get System Info Parameters command).
        - BMC management firmware (through Get Device Id command).
- The OOB management LAN configuration can be updated through the BMC.
    - This requirement is only applicable for systems that expose the BMC through the IPMI/LAN channel. The management operations are carried out over the in-band channel through the IPMI driver. The BMC must expose the following information about its own LAN configuration through the Get LAN Configuration Parameters command:
        - Indicator of whether the BMC is configured with a static IP address or if one is assigned by DHCP
        - IP Address
        - Subnet mask
        - Default Gateway
        - Primary and secondary DNS
    - The BMC must support the following operations being performed through the Set LAN Configuration Parameters command:
        - BMC can be configured with a static IP address, Subnet Mask, Default Gateway IP address, and primary and secondary DNS
        - BMC can be configured to get its IP address from the DHCP server
- The system's power state can be managed through the BMC.
    - The BMC must expose the following server system information:
        - Current power state of the server (through the Get Chassis Status command)
    - The BMC must support the following operations being performed on the server system through the Chassis Control command:
        - The server power can be turned off

- The server power can be turned on
- The server power can be reset

- The BMC prevents untrusted access to the server system.
  - The authentication mechanism used by IPMI presents a number of vulnerabilities that are exploitable in a BMC with unsecure configuration. To mitigate some of these vulnerabilities, the following configuration is present on certified BMCs:
    - BMC must not allow remote access on the LAN channel using the RAKP-none Authentication Algorithm.
    - BMC must not have an anonymous user account configured by default. If this account exists, it must be disabled.
- The system's basic inventory is exposed through the BMC.
  - The BMC must expose the following server system information:
    - Manufacturer of the server hardware (Read FRU Data command)
    - Model of the server hardware (Read FRU Data command)
    - Server SMBIOS GUID (Get System GUID command)
      - The expected format of the GUID on the wire conforms to the format described in the SMBIOS 2.8 Specification (DSP0134). That is, the GUID 00112233-4455-6677-8899-AABBCCDDEEFF is transmitted as 33 22 11 00 55 44 77 66 88 99 AA BB CC DD EE FF by the Get System GUID command.
    - Asset Tag of the server (Read FRU Data command)
    - Serial Number of the server (Read FRU Data command)
    - System Event Log time of the server (Get SEL Time command)
    - System Event Log capacity information (Get SEL Info command)
- The BMC allows remote credential management.
  - The BMC must support its Administrator password being changed through the Set User Password command. This operation is executed using the in-band channel through the IPMI driver.
- The System Event Log (SEL) can be managed through the BMC

- The SEL entries can be read (Get SEL Entry command)
- The SEL can be cleared (Clear SEL command)

Below is a list of the IPMI commands that are being used to manage BMC devices:

- Open Session
- RAKP Messages
- Set Session Privilege Level
- Get Session Info
- Close Session
- Get Device Id
- Get System GUID
- Get System Info Parameters
- Read FRU Data
- Chassis Control
- Get Chassis Status
- Get Channel Info
- Get LAN Configuration Parameters
- Set LAN Configuration Parameters
- Warm Reset
- Get SEL Entry
- Get SEL Info
- Reserve SEL
- Clear SEL
- Get SEL Time
- Get System Boot Options
- Set System Boot Options

- Get User Name
- Set User Password

Enforcement

This is an “If-Implemented” optional device requirement. This is a prerequisite device requirement for server claiming to be out-of-band manageable using the de facto IPMI standard. This requirement becomes in effect at the release of Windows Server vNext.

Send comments about this topic to Microsoft

Device.Connectivity.UsbDevices

Applies to all devices connected via USB including USB hubs. Does not apply to USB controllers.

In this topic:

- Device.Connectivity.UsbDevices.UseUsbClassOnlyForControllerOrHub
- Device.Connectivity.UsbDevices.WirelessUsbObtainsWusbLogoFromUsbif
- Device.Connectivity.UsbDevices.WirelessUsbWiMediaAlliace

Device.Connectivity.UsbDevices.DebugCompliesWithDebugSpec

USB debug device must comply with the USB2 debug device specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB devices designed for debug purposes over USB 2.0 must comply with USB2 Debug Device Functional Specification, which includes details on the device framework, commands, and additional operational requirements.

Device.Connectivity.UsbDevices.DebugCompliesWithDebugSpecUSB3

USB 3.0 debug cables must comply with the USB 3.0 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB cables designed for USB 3.0 host debugging must comply with the Universal Serial Bus 3.0 Specification, section 5.5.2.

Device.Connectivity.UsbDevices.DeviceAttachLessThan100ms

USB device that signals device-attach must respond after at least 100 ms.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

When the USB device has signaled device-attach, the operating system provides a debounce interval of 100ms. The device must respond at the end of that interval. This is described in USB Specification,

Revision2.0, Section 7.1.7.3. This requirement ensures that the electrical and mechanical connections are stable before the attached device is reset.

Device.Connectivity.UsbDevices.FunctionSuspendSelectiveSuspend

USB 3.0 devices must correctly implement Function Suspend and Selective Suspend.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Any function that is in a suspend state before a device is selectively suspended remains in the function suspend state when the device is resumed from the selective suspend state. Devices must not place all device functions in the function suspend state. Devices must report the selective suspend state.

SuperSpeed devices ignore the DEVICE_REMOTE_WAKEUP feature selector.

When all functions of a SuperSpeed device are in the function suspend state and the PORT_U2_TIMEOUT field is programmed to 0xFF, the device initiates U2 after 10 milliseconds (ms) of link inactivity. For more information, see section 9.2 of the USB 3.0 Specification.

Devices that are resumed from the selective suspend state retain a minimum set of device state information as specified in section 9.2.5.2 of the USB 3.0 Specification.

Device.Connectivity.UsbDevices.InternalDevicesMustSupportSuspend

All internally connected USB devices must go to Selective Suspend after periods of inactivity.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All internally connected USB devices must be capable of entering the suspend state after device drivers for those devices initiate the USB Selective Suspend process. Third-party drivers for internal devices on applicable systems must implement USB Selective Suspend.

Devices belonging to these device classes can opt out of supporting USB Selective Suspend:

- Printers
- Scanners
- Fax

Device.Connectivity.UsbDevices.IsochronousDeviceAndDriver

Isochronous USB device and driver requirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

1. ISO USB device and driver provide multiple alternate settings to support maximum flexibility of hardware interface options:

If any alternate setting consumes isochronous bandwidth, devices and drivers must provide multiple alternate settings for each interface.

2. USB device and driver do not use isochronous bandwidth for alternate setting 0:

Devices and drivers must not use isochronous bandwidth for alternate setting 0. Devices must consume bandwidth only when they are in use.

3.USB isochronous full-speed or high-speed device that uses more than 50 percent of USB bus bandwidth provides alternate settings:

If a USB isochronous full-speed or high-speed device uses more than 50 percent of USB bus bandwidth, it must provide alternative settings that allow the device to switch to a setting that uses less than 50 percent of the bus bandwidth and operate as a device of that particular class (ex. if it is a camera then it must continue to stream video), even though it may be in a lower quality mode.

Devices with attached host controllers are exempt from this requirement.

4. USB device with interfaces containing isochronous endpoints has at least one alternative interface for low bandwidth scenarios:

USB device must provide at least one alternative interface for low bandwidth as described in USB Specification, Revision 2.0 or later, Section 9.6.5.

Design Notes:

See section 9.6.5 in the Universal Serial Bus Specification, Revision 2.0.

If two or more devices are connected that use more than 50 percent of the bus bandwidth and do not provide alternate settings, only one of the devices works at a time.

See USB Specification, Revision 2.0 or later, Sections 5.6 and 5.7.

Device.Connectivity.UsbDevices.MsOsContainerId

USB devices that implement the Microsoft OS Container ID descriptor must implement it correctly.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

If a multifunction USB device implements the Microsoft® operating system **ContainerID** descriptor, the device does this in the Microsoft operating system feature descriptor.

The Microsoft operating system **ContainerID** descriptor allows Windows® to correctly detect multifunction devices. The descriptor provides a way for all the device nodes to appear as one physical object in the **Devices and Printers** user interface (UI).

Device.Connectivity.UsbDevices.MustBeFunctionalAfterResume

Attached USB devices must be functional after resuming from system power states.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Devices not entering a timely ready state will be marked code 10 or other by the system. Certain classes of devices do not properly respond to system events, such as resume, and require upper driver or expect precise boot timings in order to function properly. A device must be able to function without a port reset upon resume, but must also remain functional if a reset does occur.

A device must be in the attached state (USB Specification 2.0, section 9.1) to be configured and the device must be in the configured state before its functions maybe used (aka, the device is useable). This is per the USB spec 2.0 as in sections 9.1 and 9.2.6.2 "After a port is reset or resumed, the USB System Software is expected to provide a "recovery" interval of 10 ms before the device attached to the port is expected to respond to data transfers. The device may ignore any data transfers during the recovery interval."

Clarification:

Devices must be functional after resuming from system power states whether a port reset is issued or not.

Device.Connectivity.UsbDevices.MustNotDisconnectDuringSuspend

USB devices must not disconnect from the upstream port while going to or resuming from selective suspend.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB devices must not disconnect from the upstream port during the selective suspend process.

To test this requirement, we will cause the device to go into the selective suspend state and then resume the device. During this process, we will observe the port status bits of the upstream port and verify that the device does not disconnect during this time. We will repeat this process serveral times.

Device.Connectivity.UsbDevices.MustResumeWithoutForcedReset

All USB devices work properly upon resume from sleep, hibernation, or restart without a forced reset of the USB host controller.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All USB devices work properly upon resume from sleep, hibernation, or restart without a forced reset of the USB host controller.

Design Notes:

Registry key ForceHCResetOnResume documented at the KB below is not needed for devices to function properly upon resume in Windows 7: http://support.microsoft.com/kb/928631.

Note that a known set of currently existing devices do require a forced reset upon resume, these devices should be covered in a list kept by the OS which will reset these devices upon resume. The goal of this requirement is to ensure that this list of devices that must be reset to appear after resume does not grow and that devices can properly handle sleep state transitions without being reset.

A reset of the entire USB Host Controller results in significantly increased time that it takes for all USB devices to become available after system resume since there could be only one device at address 0 at a time, this enumeration has to be serialized for all USB devices on the bus. We have also seen that resetting the host controller can lead to an illegal SE1 signal state on some host controllers, which in turn can cause some USB devices to hang or drop off the bus. Moreover, devices cannot maintain any private state across sleep resume as that state will be lost on reset.

Device.Connectivity.UsbDevices.MustSignalAttachWithin500ms

Devices must signal attach within 500 ms after the system resumes.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

After the system resumes from sleep, the hub driver will fetch the status of the port to which the device is connected. If the device does not show as connected, the hub driver will wait 500 milliseconds (ms) before the hub driver queries the port status again. If the device appears as connected within that time, the hub will not need to re-enumerate the device for Plug and Play, which will result in a better user experience. This requirement already exists for bus-powered devices in the USB specification. The requirement will now also apply to self-powered devices.

Device.Connectivity.UsbDevices.MustSupportSuspend

All bus powered USB devices must support USB Suspend after periods of inactivity.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The device driver for a bus-powered device initiates the USB Selective Suspend process. The device can enter the suspend state from any powered state. The device must begin the transition to the suspend state after it sees a constant idle state on its upstream facing bus lines for more than 3.0 ms. In suspend state, the device must only draw suspend current from the bus after no more than 10 ms of bus inactivity on all its ports, as described in the USB Specification, Revision 2.0, Sections 7.1.7.6, 6 9.1.1.6 and 9.2.6.2.

Clarification about USB Selective Suspend in embedded USB devices can be found in the following requirement: Device.Connectivity.UsbDevices.InternalDevicesMustSupportSuspend.

Clarification about USB Selective Suspend in Windows RT systems can be found in the following requirements: Device.Connectivity.UsbDevices.MustSupportSuspendOnRT.

Device.Connectivity.UsbDevices.RespondAllStringRequests

A USB device must respond to all string requests that the host sends to indexes.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB devices must respond accordingly to string requests that the host sends. Devices must stall if no string is stored at the index being queried or if a request error exists. Devices must not reset themselves or stop functioning. This is described in USB Specification, Revision 2.0 or later, Section 9.6.

Device.Connectivity.UsbDevices.ResponsesLimitedByWlengthField

USB device responses to host requests are limited in size by the wLength field.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All USB device requests contain a wLength field. Responses by the USB device to host requests must be of size <= wLength field of the device request as defined in the USB Specification, Revision1.1 or later, Section 9.3.5.

Device.Connectivity.UsbDevices.SerialNumbers

USB serial numbers are implemented for specific device classes and are unique across specific device models.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB serial numbers must be implemented for the following device classes:

- Bluetooth (Class Code 0xE0, SubClass 0x01, Protocol 0x01)
- Communication device class (Class Code 0x02)
- Mass storage (Class Code 0x08)
- Scanning/imaging (Class Code 0x06)
- Printing (Class Code 0x07)
- Host Wire Adapters and Device Wire Adapters (Class Code 0xE0, subclass 02)

USB serial numbers are optional for all other device classes. Additionally, if serial numbers are implemented on the device's model, all devices of the same model must have unique serial numbers.

Design Notes:

For more information on USB device class details, see "Defined 1.0 Class Codes" at: http://go.microsoft.com/fwlink/?LinkId=40497.

For more information on implementation of serial numbers, see USB Specification, Revision 2.0 or later, Section 9.6.

Device.Connectivity.UsbDevices.SerialNumbersUseValidCharacters

A USB device that implements manufacturer-defined serial numbers must contain valid characters.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A USB serial number must be a string that contains a manufacturer-determined ID composed of valid characters. Valid characters are defined in the Windows Driver Kit, "USB_DEVICE_DESCRIPTOR."

Device.Connectivity.UsbDevices.SuperSpeedOnConnectViaUsb3Port

If upstream SuperSpeed termination is on, devices must always connect on the USB 3.0 port and never connect on the USB 2.0 port.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

In a USB 3.0 hub, two downstream ports, a USB 2.0 port and a Superspeed port share the same connector. When a SuperSpeed (that is, non-hub) device is plugged into such a connector, the device must always connect on the SuperSpeed port. The device must never connect on the USB 2.0 port.

To test this requirement, the software will verify that the USB 3.0 port status bits show a connected device and that the USB 2.0 port status bits do not show a connected device.

For a definition of "connect", see section 2 of the USB 3.0 Specification under "connected".

Device.Connectivity.UsbDevices.TestedUsingMicrosoftUsbStack

USB devices must be tested with Microsoft's xHCI Stack installed.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All USB Devices (Low, Full, High, and Super Speed devices) must be tested with Microsoft's Extensible Host Controller Interface (xHCI) Stack installed and enabled on a Windows system.

Note: During USB-IF self-testing a specific USB Test Stack is installed for testing purposes, this is expected and acceptable.

Device.Connectivity.UsbDevices.UsbifCertification

USB devices must either pass USB IF tests or be USB IF certified.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB devices must pass USB Implementers Forum (IF) tests or be USB-IF certified.

See white paper on Windows Logo Kit USB-IF Testing

At http://www.microsoft.com/whdc/connect/usb/wlk-usb-if-testing.mspx

Device.Connectivity.UsbDevices.UseUsbClassOnlyForControllerOrHub

Third-party INF files include the class "USB" only if the device is a USB host controller, a root, or an external hub.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Class USB is often used incorrectly for devices that do not have a predefined class. For example, a USB mouse uses class HID, whereas a USB smartcard uses class smartcard reader. Class USB is reserved for host controllers and root or external USB hubs. If the vendor has a device that has no Windows-defined class but uses USB as the bus, it must define its own class or class GUID. The setup class associated with the type of USB device, not with the bus type, must be used. The setup class "USB" (ClassGuid = {36fc9e60-c465-11cf-8056-444553540000}) is reserved for USB host controllers and root or external USB hubs. It must not be used for other device categories.

Design Notes:

Microsoft provides system-defined setup classes for most device types. System-defined setup class GUIDs are defined in the Windows Driver Kit, "Devguid.h."

If you choose the wrong class, the device appears in an incorrect location in Device Manager and in the Windows Vista UI. Using this class incorrectly may cause the device driver to fail hardware compatibility testing.

For a list of Windows class GUIDs, see the Windows Driver Kit, "System-Supplied Device Setup Classes" at: http://msdn.microsoft.com/en-us/library/ff553419(VS.85).aspx.

Device.Connectivity.UsbDevices.WirelessUsbObtainsWusbLogoFromUsbif

Wireless USB device or host must obtain a Certified Wireless USB logo from the USB-IF.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All Wireless USB devices must get a Certified Wireless USB Logo from the USB-IF.

Device.Connectivity.UsbDevices.WirelessUsbWiMediaAlliace

Certified Wireless USB device or host must pass all required WiMedia Alliance compliance tests.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Wireless USB device must pass WiMedia Alliance radio compliance tests.

Send comments about this topic to Microsoft

Device.Connectivity.UsbHub

Requirements that apply only to USB Hubs

In this topic:

Device.Connectivity.UsbHub.IdentifyNumOfUserAccessiblePorts

A USB hub must correctly identify and report the number of ports that the user can access.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The USB hub must include details in its hub descriptor that provide the operating system with an accurate count of the number of downstream-facing ports that the hub supports and that are exposed to the user. See USB Specification, Revision 2.0, Section11.23, and USB 3.0 Specification, Section 10.14. Root hubs are exempt from this requirement.

Device.Connectivity.UsbHub.SupportSuspend

USB hubs must support the selective suspend state, and downstream devices must not drop off the bus when the hub resumes from selective suspend.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB hubs must support the selective suspend state, as stated in both the USB Specification and other compatibility program requirements. After a hub is resumed from the selective suspend state, all devices that were attached downstream of the hub, and that were not removed while the hub was suspended, must be present.

Device.Connectivity.UsbHub.Usb3HubCompliesWithUsb3Spec

USB 3.0 hubs are compliant with the USB 3.0 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB 3.0 hubs must be compliant with the Universal Serial Bus (USB) 3.0 specification.

USB 3.0 hubs must:

- Pass the USB-IF interoperability tests

- Pass the USB 3.0 Hub compliance test suite
- Pass the USB 3.0 CV test

Device.Connectivity.UsbHub.Usb3ReportPortStatusBitsCorrectly

USB 3.0 hubs must always report the port status bits correctly as per the USB 3.0 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

In the current stack, a number of invalid port status bit combinations that the hub reports are ignored. Any invalid combination of port status bits will be treated as an error. In particular, checks will follow these actions:

- Resetting a port
- Suspending and resuming a port
- System resume

A hub should not report spurious change interrupts. A hub should complete the port status interrupt transfer without reporting changes.

Send comments about this topic to Microsoft

Device.Connectivity.WSD

In this topic:

Device.Connectivity.WSD.DPWS

Devices which use or interact with the Web Services on Devices API (WSDAPI) comply with Device Profiles for Web Services (DPWS) specification

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Devices which plan to use or interact with Microsoft Windows' implementation of DPWS, the Web Services on Devices API (WSDAPI), must implement the DPWS specification themselves. (WSDAPI)

Design Notes:

DPWS Specification available at

http://go.microsoft.com/fwlink/?LinkId=109231

Device.Connectivity.WSD.DPWSExtensibility

Devices Profile for Web Services Devices must accept messages that contain extensibility sections, and process the messages as appropriate.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Devices Profile for Web Services (DPWS) devices must accept messages where the XML has been extended. If the device understands the content in the extensible section, it may process it.

Design Notes:

DPWS Specification available at

http://go.microsoft.com/fwlink/?LinkId=109231

Device.Connectivity.WSD.MetadataExchange

Devices Profile for Web Services (DPWS) Devices support metadata exchange

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

DPWS Devices which interact with the Web Services on Devices API (WSDAPI) must support metadata exchange as defined in the metadata exchange specification.

Design Notes:

Metadata Exchange specification can be obtained at http://go.microsoft.com/fwlink/?LinkId=109248

Device.Connectivity.WSD.MetadataValid

Devices which interact with the Web Services on Devices (WSDAPI) produce metadata that conforms to the Devices Profile for Web Services

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Devices which interact with WSDAPI must populate the Metadata as defined in the Device Profile for Web Services Specification of February 2006.

Design Notes:

The Device Profile for Web Services Specification of February 2006 is available at http://go.microsoft.com/fwlink/?LinkId=109231

Device.Connectivity.WSD.Schema

A network-enabled device that implements Devices Profile for Web Services (DPWS) must adhere to the protocol and schema.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

A network-enabled device that implements Devices Profile for Web Services (DPWS) must adhere to the Devices Profile for Web Services as described by the schema.

The device must also reference the namespace URI as described in The Devices Profile for Web Service specification.

A device the implements DPWS must adhere to the Web Services Description Language (WSDL) associated with the logo device class. The WSDL defines services as collections of network endpoints, or ports. WSDL specification provides an XML format for documents for this purpose. Devices must implement the WSDL version 1.1.

Design Notes:

See the Web Services Description Language (WSDL) Version 1.1 at http://www.w3.org/TR/2001/NOTE-wsdl-20010315

See the Devices Profile for Web Services schema at http://schemas.xmlsoap.org/ws/2006/02/devprof/devicesprofile.xsd.

See the Devices Profile for Web Service specification at http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf.

Additional information can be found in the Windows Rally Development Kit at http://go.microsoft.com/fwlink/?LinkId=109368.

Device.Connectivity.WSD.WSDiscovery

Devices Profile for Web Services (DPWS) Devices interacting with the Web Services on Devices API (WSDAPI) implement WS-Discovery

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

DPWS Devices must implement WS-Discovery to work with WSDAPI.

Design Notes:

WS-Discovery specification can be obtained at http://go.microsoft.com/fwlink/?LinkId=109247

Send comments about this topic to Microsoft

Device.DevFund.CDA

Custom Driver Access for privileged application usage.

In this topic:

Device.DevFund.CDA.Application

Custom Driver Access

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a device driver supports a privileged app performing Custom Driver Access, it must declare a restricted interface.

By declaring a restricted interface, the following requirements must be met:

- Assert that the device io control interfaces provided by this device driver are intended to be accessed by a privileged app running in an app container that accesses hardware functionality using CreateDeviceAccessInstance() and IDeviceIoControl() on Windows 10.
- The restricted interface cannot be opened directly from an app container.

A device driver declares an interface is restricted by setting the DEVPKEY_DeviceInterface_Restricted property to true on that interface.

Send comments about this topic to Microsoft

Device.Devfund.DeviceGuard

All kernel drivers must be built to be compatible with Device Guard.

In this topic:

Device.DevFund.DeviceGuard.DriverCompatibility

Applies to	Windows 10 x64 Windows Server 2016 Technical Preview x64

Description

Windows 10 has a new feature called Device Guard that gives organizations the ability to lock down devices in a way that provides advanced malware protection against new and unknown malware variants as well as Advanced Persistent Threats (APTs). Device Guard can use hardware technology and virtualization to isolate the Code Integrity (CI) decision-making function from the rest of the Windows operating system. When using virtualization-based security to isolate Code Integrity, the only way kernel memory can become executable is through a Code Integrity verification. This means that kernel memory pages can never be Writable and Executable (W+X) and executable code cannot be directly modified.

Details are available in the Windows Hardware Certification blog.

Send comments about this topic to Microsoft

Device.DevFund.DriverFramework.KMDF

Driver framework requirements for KMDF

In this topic:

Device.DevFund.DriverFramework.KMDF.Reliability

Kernel Mode Driver Framework (KMDF) drivers must be architected to maximize reliability and stability and do not "leak" resources such as memory and KMDF objects.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Kernel-Mode Driver Framework (KMDF) drivers must use pool memory responsibly. Handles that the drivers pass to device driver interfaces (DDIs) must conform to the pattern of the parameter. The state of the drivers must be consistent with the use of **WDFREQUEST** objects and **WDFQUEUE** objects.

Event callback functions that the driver registers must adhere to interrupt request level (IRQL) restrictions.

Design Notes:

For more information about developing drivers that meet this requirement, visit the following websites:

http://msdn.microsoft.com/en-us/library/aa973499.aspx

http://www.microsoft.com/whdc/driver/wdf/KMDF.mspx

The following tools can be enabled to validate this requirement for all KMDF drivers:

- Windows® Driver Foundation (WDF) Verifier.
- Handle tracking. Handle tracking will be enabled on all KMDF objects.

- Enhanced Verifier for Framework 1.9 KMDF drivers. Enhanced Verifier is new for Framework 1.9. This tool can be enabled by using the EnhancedVerifierOptions registry value. To enable Enhanced Verifier, set the following registry values for the driver's Parameters\Wdf key:

HKLM\System\CurrentControlSet\Services\\Parameters\Wdf

EnhancedVerifierOptions REG_DWORD 1

VerifierOn REG_DWORD 1

TrackHandles MULTI_SZ *

- Driver Verifier. To enable Driver Verifier, use the following command:

Verifier /flags 0xfb /driver

This command will run the KMDF driver under Driver Verifier with all flags set except the Low Resource Simulation flag. For more information about Driver Verifier, visit the following website:

http://msdn.microsoft.com/en-us/library/ff545448.aspx

In the Windows Hardware Lab Kit, the WDF Test can be run to validate this requirement.

Device.DevFund.DriverFramework.KMDF.WDFProperINF

Windows Driver Framework (WDF) driver INF files must be properly structured.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- All information (INF) files in Windows Driver Foundation (WDF) driver packages must call WDF-specific sections properly. Correctly structured INF sections help to ensure that the driver will be installed properly. However, even when a driver is installed, a poorly or wrongly configured section can cause unpredictable problems during the operation of the driver or device. These problems can be prevented by following the guidelines for WDF INF settings.

  To meet this requirement, all WDF INF files must have the following:

- A coinstaller section, as follows:

```
[DDInstall.Coinstallers]
CopyFiles=
AddReg=
```

```
```

- A WDF section, as follows:

  For Kernel-Mode Driver Framework (KMDF) drivers:

```
[DDInstall.Wdf]
KmdfService= <ServiceName>, <Kmdf_Install>
[Kmdf_Install]
KmdfLibraryVersion=
```

  For User-Mode Driver Framework (UMDF) drivers:

```
[DDInstall.Wdf]
UmdfService=<ServiceName>,<Umdf_Install>
UmdfServiceOrder=
UmdfDispatcher [Only for USB Drivers and Drivers with file handle I/O
targets]=
UmdfImpersonationLevel[optional]=

[Umdf_Install]
UmdfLibraryVersion=
DriverCLSID=
ServiceBinary=
```

- All UMDF driver INF files must have a WUDFRD service installation section, as follows:

```
[WUDFRD_ServiceInstall]
DisplayName = "Windows Driver Foundation - User-mode Driver Framework
Reflector"
ServiceType = 1
StartType = 3
ErrorControl = 1
ServiceBinary = %12%\WUDFRd.sys
LoadOrderGroup = Base
```

- All WDF drivers that use a WinUSB driver must have the following service installation settings:

```
[WinUsb_ServiceInstall]
DisplayName = "WinUSB Driver"
ServiceType = 1
StartType = 3
```

```
ErrorControl = 1
ServiceBinary = %12%\WinUSB.sys
```

- Service names, hardware IDs (HWIDs), display names, and UMDF class identifiers (CLSIDs) cannot be pasted from WDF samples.

Design Notes:

For more information about WDF-specific INF settings, visit the following websites:

http://www.microsoft.com/whdc/driver/wdf/wdfbook.mspx

http://msdn.microsoft.com/en-us/library/ff560526.aspx

http://msdn.microsoft.com/en-us/library/ff560526.aspx

Send comments about this topic to Microsoft

Device.DevFund.DriverFramework.UMDF

Driver framework requirements for UMDF

In this topic:

Device.DevFund.DriverFramework.UMDF.Reliability

User Mode Driver Framework (UMDF) drivers must be secure, stable, reliable, and not have application compatibility issues.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To help ensure that all User-Mode Driver Framework (UMDF) drivers meet security standards, are stable and reliable, and do not have application compatibility issues, drivers must not have any object leaks.

Object leaks can be diagnosed by enabling object tracking. If a memory leak occurs, set the Reference Count Tracking setting to "On." This setting logs the history for "add of reference" and "release of reference" counts.

These features can be set to "On" by using the following registry values:

HKLM\Software\Microsoft\WindowsNT\CurrentVersion\WUDF\Services\{193a1820-d9ac-4997-8c55-be817523f6aa}

TrackObjects REG_DWORD 1

TrackRefCounts REG_DWORD 1

UMDF drivers must also meet the following requirements:

- The drivers must not attempt to use invalid handles.
- The drivers must use critical sections and file locks properly. The primary purpose of the locks test is to help ensure that the application properly uses critical sections.
- The drivers must not cause heap memory corruption.
- The drivers must correctly use virtual address space manipulation functions, such as VirtualAlloc, VirtualFree, and MapViewOfFile.
- The drivers must not hide access violations by using structured exception handling.
- The drivers must correctly use thread local storage functions.
- The drivers must use COM correctly.

Partners can verify that the drivers meet these requirement by enabling Microsoft® Application Verifier's handles, locks, heaps, memory, exceptions, and Transport Layer Security (TLS) settings for the UMDF host process (that is, WUDFHost.exe) during driver development and testing.

For more information, see the Developing Drivers with the Windows Driver Foundation book at the following website:

http://www.microsoft.com/whdc/driver/wdf/wdfbook.mspx.

Design Notes:

Application Verifier will help check UMDF drivers extensively to help ensure stable and reliable drivers.

For all UMDF drivers, Application Verifier will be enabled when driver reliability tests are executed.

Application Verifier can be downloaded from the following Microsoft website:

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=c4a25ab9-649d-4a1b-b4a7-c9d8b095df18.

To enable Application Verifier for WUDFHost.exe, run the following command:

appverif -enable handles locks heaps memory COM exceptions TLS -for WUDFHost.exe.

For more information about UMDF, visit the following website:
http://www.microsoft.com/whdc/driver/wdf/UMDF.mspx.

Device.DevFund.DriverFramework.UMDF.WDFProperINF

Windows Driver Framework (WDF) driver INF files must be properly structured.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All information (INF) files in Windows Driver Foundation (WDF) driver packages must call WDF-specific sections properly. Correctly structured INF sections help ensure that the driver will be installed properly. However, even when a driver is installed, a poorly or wrongly configured section can cause unpredictable problems during the operation of the driver or device. These problems can be prevented by following the guidelines for WDF INF settings.

To meet this requirement, all WDF INF files must have the following:

- A coinstaller section, as follows: [DDInstall.Coinstallers]

```
[DDInstall.Coinstallers]
CopyFiles=
AddReg=
```

- A WDF section, as follows:

  For Kernel-Mode Driver Framework (KMDF) drivers:

```
[DDInstall.Wdf]
KmdfService= <ServiceName>, <Kmdf Install>
[Kmdf_Install]
KmdfLibraryVersion=
```

  For User-Mode Driver Framework (UMDF) drivers:

```
[DDInstall.Wdf]
UmdfService=<ServiceName>,<Umdf_Install>
UmdfServiceOrder=
UmdfDispatcher [Only for USB Drivers and Drivers with file handle I/O
targets]=
UmdfImpersonationLevel[optional]=

[Umdf_Install]
UmdfLibraryVersion=
DriverCLSID=
ServiceBinary=
```

- All UMDF driver INF files must have a WUDFRD service installation section, as follows:

```
[WUDFRD_ServiceInstall]
DisplayName = "Windows Driver Foundation - User-mode Driver Framework
Reflector"
ServiceType = 1
StartType = 3
ErrorControl = 1
ServiceBinary = %12%\WUDFRd.sys
LoadOrderGroup = Base
```

- All WDF drivers that use a WinUSB driver must have the following service installation settings:

```
[WinUsb_ServiceInstall]
DisplayName = "WinUSB Driver"
ServiceType = 1
StartType = 3
ErrorControl = 1
ServiceBinary = %12%\WinUSB.sys
```

- Service names, hardware IDs (HWIDs), display names, and UMDF class identifiers (CLSIDs) cannot be pasted from WDF samples.

Design Notes:

For more information about WDF-specific INF settings, visit the following websites:

http://www.microsoft.com/whdc/driver/wdf/wdfbook.mspx

http://msdn.microsoft.com/en-us/library/ff560526.aspx

http://msdn.microsoft.com/en-us/library/ff560526.aspx

Send comments about this topic to Microsoft

Device.DevFund.Firmware

Driver package requirements for firmware update package

In this topic:

Device.DevFund.Firmware.UpdateDriverPackage

These requirements apply to any firmware update driver package that is submitted to Microsoft for approval and signing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

In addition to standard driver requirements, the following requirements apply to firmware update driver package:

- A firmware package must payload the update for only one resource.
- A firmware package must be configurable (see Device.DevFund.INF.* for more details).
- After a successful firmware upgrade, the firmware version in the .INF file of the driver package, the resource version (in ESRT), and the last attempted version (in ESRT) for that resource must match.
- The name of the binary file in the firmware package must not conflict with any of the previous firmware versions.
- A successful firmware upgrade must not reduce or eliminate the functionality of any devices in the system.

Send comments about this topic to Microsoft

Device.DevFund.INF

INF restictions

In this topic:

- Device.DevFund.INF.FileOrRegistryModification
- Device.DevFund.INF.InstallManagement
- Device.DevFund.INF.LegacySyntax
- Device.DevFund.INF.TargetOSVersion

Device.DevFund.INF.AddReg

When using an AddReg directive, each AddReg entry must specify HKR as the registry root.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

HKR (meaning "relative root") is the only registry root identifier that can be referenced in an AddReg section in an INF file. Other root identifiers, including HKCR, HKCU, HKLM, and HKU, are restricted from use in an AddReg section. The AddReg directive is intended to be used for device installation and configuration purposes only.

Design Notes:

All registry keys declared in an AddReg section of an INF file must use the relative root identifier (HKR) as the registry root value, unless an explicit exception exists as outlined in this requirement.

The following example shows the registration of a COM object using AddReg directives. Building on this example, it is possible to customize all of the object's parameters:

```
[COMobj.AddReg]
HKCR,CLSID\{<CLSID>},,,"<MFT DLL description>"
HKCR,CLSID\{<CLSID>}\InprocServer32,,%REG_EXPAND_SZ%,"%%SystemRoot%%\Syst
em32\mftxyz.dll"
HKCR,CLSID\{<CLSID>}\InprocServer32,ThreadingModel,,"Both"
```

A complete list of COM registry entries with details on their use can be found in the MSDN at: http://msdn.microsoft.com/en-us/library/ms694355(v=vs.85).aspx.

The following example shows the registration of an MFT filter using AddReg directives:

```
[MFT.AddReg]
HKCR,CLSID\{<CLSID>},,,"<MFT DLL description>"
HKCR,CLSID\{<CLSID>}\InprocServer32,,%REG_EXPAND_SZ%,"%%SystemRoot%%\Syst
em32\mftxyz.dll"
HKCR,CLSID\{<CLSID>}\InprocServer32,ThreadingModel,,"Both"
HKCR,MediaFoundation\Transforms\<CLSID>,InputTypes,%REG_BINARY%,76,45,87,
2d,5e,23,...
```

```
HKCR,MediaFoundation\Transforms\<CLSID>,OutputTypes,%REG_BINARY%,22,5e,23
,46,43,10,...
HKCR,MediaFoundation\Transforms\<CLSID>,,%REG_SZ%,"MFT Friendly Name"
HKCR,MediaFoundation\Transforms\<CLSID>,MFTFlags,%REG_DWORD%, 0x00000004
HKCR,MediaFoundation\Transforms\<CLSID>,Attributes,REG_BINARY%, 41,46,4d,
HKCR,MediaFoundation\Transforms\Categories\<MFTCategoryGUID>\<CLSID>
HKLM,SOFTWARE\Microsoft\Windows Media
Foundation\ByteStreamHandlers\audio/xyz,<CLSID>,,"XYZ Stream Handler"
```

Additionally, when registering a DECODE or ENCODE HMFT, one of the following registry keys must also be set:

```
DECODE HMFT
HKLM,SOFTWARE\Microsoft\Windows Media
Foundation\HardwareMFT,EnableDecoders, %REG_DWORD%, 1

 ENCODE HMFT
HKLM,SOFTWARE\Microsoft\Windows Media
Foundation\HardwareMFT,EnableEncoders, %REG_DWORD%, 1
```

More details on MFTs can be found in the MSDN at: http://msdn.microsoft.com/en-us/library/windows/desktop/ms703138(v=vs.85).aspx.

Additional Information

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) and Windows Server Technical Preview.It will be required in the future for those architectures.Note that there are some exceptions to this requirement to accommodate the registration of Component Object Model (COM) objects and Media Foundation Transforms (MFT) using the AddReg directive. Refer to the Design Notes section of this requirement for additional details.

Device.DevFund.INF.AddService

INF files can only install driver-related services.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

An INF AddService directive can only reference services that are driver related. Services that are not driver related, such as a Microsoft Win32 service, cannot be referenced or installed using an INF file.

Design Notes:

An INF AddService directive service-install-section may only specify a ServiceType type-code of the following:

- SERVICE_DRIVER
- SERVICE_KERNEL_DRIVER
- SERVICE_FILE_SYSTEM_DRIVER

Additional Information

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.ClassInstall32

INF files must not define a custom class installer within a ClassInstall32 section.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

An INF file may not specify a custom class installer within a ClassInstall32 section. Therefore, a driver package cannot execute a custom class installer during device installation.

Design Notes:

Developers should use one of the existing inbox device setup classes for their device. If it is necessary to define a new device setup class, the new setup class cannot employ a class installer as part of the device installation process. The following example shows an INF ClassInstall32 section, which defines a custom class installer and therefore fails this requirement.

```
[ClassInstall32.ntx86]          ; Declare a ClassInstall32 section for the x86
                                ; architecture.
AddReg=SetupClassAddReg    ; Reference to the ClassInstall32 AddReg section.
; Place additional class specific directives here
[SetupClassAddReg]              ; Declare a class specific AddReg section.
; Device class specific AddReg entries appear here.
; The next line defines the class installer that will be executed when
; installing devices of this device-class type. Defining a registry entry
```

; of this type is no longer supported and the driver package fails to meet

; this device fundamental requirement.

[HKR,,Installer32,,"class-installer.dll,class-entry-point"]

Additional Information

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.ComplexDeviceMatching

INF directives related to complex device matching logic are not supported.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

INF files will not support complex device matching logic. Specifically, the capability to specify a DeviceID for a device that should not be installed, when a matching HardwareID or CompatibleID exists in the DDInstall section, will not be supported.

Design Notes:

The following INF directive may not be referenced in an INF file:

- ExcludeID

Additional Information

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.DDInstall.CoInstallers

INF files must not reference any co-installers within a DDInstall.CoInstallers section.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

An INF file may not reference any co-installers within a DDInstall.CoInstallers section. Therefore, a driver package cannot execute any co-installers during device installation.

Design Notes:

Execution of co-installers is prohibited during device installation. The following examples show the registration of a device-specific co-installer and a device-class co-installer. Both types of co-installers are not permitted in an INF file and inclusion will result in failure to meet the requirement.

Device-specific co-installer example:

```
; Registering one or more device-specific co-installers requires adding
; adding a REG_MULTI_SZ value using an AddReg directive. The following
; shows the general form for registering a device-specific co-installer.
;  :
;  :
[DestinationDirs]       ; Destination dir for the co-installer dll
XxxCopyFilesSection = 11          ; DIRID_for %WINDIR%\System32 dir
                        ; Xxx = driver or device prefix
;  :
;  :
[XxxInstall.OS-platform.CoInstallers]      ; Define co-installers section
CopyFiles = XxxCopyFilesSection           ; Copy files directive
AddReg = Xxx.OS-platform.CoInstallers_AddReg       ; Add registry directive
[XxxCopyFilesSection]           ; Define the co-installer copy files
XxxCoInstall.dll                ; section
[Xxx.OS-platform.CoInstallers_AddReg]      ; Define the co-installer AddReg
                                ; section
; The next line defines the co-installer that will be executed when
; installing this device. Defining a registry entry of this type is no
; longer supported and the driver package fails to meet this device
; fundamental requirement.
HKR,,CoInstallers32,0x00010000,"XxxCoInstall.dll, \
  XxxCoInstallEntryPoint"
```

Device-class co-installer example:

```
[Xxx.OS-platform.CoInstallers_AddReg]      ; Define the co-installer AddReg
                                ; section
```

; Similar format to the device-specific co-installer example, except the

; registry location is under HKLM. The next line defines the co-installer

; executed after any installation operations complete for the given device

; setup class GUID. Defining a registry entry of this type is no

; longer supported and the driver package fails to meet this device

; fundamental requirement.

**HKLM,System\CurrentControlSet\Control\CoDeviceInstallers, **

{SetupClassGUID}, 0x00010008,"DevClssCoInst.dll[,DevClssEntryPoint]"

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.DeviceConfigOnly

INF files cannot reference INF directives that are not directly related to the configuration of a device.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

INF directives that provide configuration functionality beyond what is necessary to configure device hardware are no longer supported. The INF file and all supporting files in the driver package must be used only for device installation and configuration.

Design Notes:

The following INF directives may not be referenced in an INF file:

- RegisterDlls
- UnregisterDlls
- ProfileItems
- UpdateInis
- UpdateIniFields
- Ini2Reg

Note that while the RegisterDlls directive can no longer be declared in an INF file, it is still possible to register Component Object Model (COM) and Media Foundation Transform (MFT) objects from an INF file using the AddReg directive. The AddReg directive allows the declaration of COM/MFT registration keys under the HKLM registry hive. For information on the use of the AddReg directive for this purpose, refer to the Device.DevFund.INF.AddReg Windows Hardware Certification requirement.

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.DeviceResourceConfig

INF based device resource configuration and non-PnP related configuration cannot be performed within an INF file.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

INF files cannot be used to perform device resource configuration or non-PnP related configuration tasks. Serveral INF directives and sections are no longer supported.

Design Notes:

The following INF sections and directives cannot be referenced in an INF file:

- [DDInstall.LogConfigOverride] section
- LogConfig
- [DDInstall.FactDef] section

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.FileCopyRestriction

INF based file copy restrictions

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

File copy destination locations are limited to prevent driver packages from installing drivers in inappropriate locations on the system.

Design Notes:

When using the CopyFiles directive, the destination directory specified for a file must be one of the following DIRID values:

- 11 (corresponds to the %WINDIR%\System32 directory)
- 12 (corresponds to the %WINDIR%\System32\Drivers directory)

Only these destination directories expressed as the appropriate DIRID will be a valid copy file location.

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.FileOrRegistryModification

Deleting or modifying existing files, registry entries, and/or services is not allowed from within an INF file.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

INF file directives that delete or modify registry entries, services, and files are no longer supported.

Design Notes:

The following INF directives may not be referenced in an INF file:

- DelReg
- DelService
- DelPropertry

- BitReg
- DelFiles
- RenFiles

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.InstallManagement

Management of files installed using an INF file is restricted to the system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Any files that are installed onto the system using an INF file are managed exclusively by Windows. Plug and Play (PnP) prevents applications from directly modifying the files that are referenced in the INF.

Design Notes:

An INF file must include the PnpLockDown directive set to value 1 in the [Version] section. This would appear as follows in the INF file:

[Version]

; Other Version section directives here.

PnpLockDown=1

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.LegacySyntax

Legacy service configuration cannot be performed within an INF file.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

Service configuration using legacy INF syntax is no longer supported.

Design Notes:

The following INF service install section directive may not be referenced in an INF file:

- LoadOrderGroup

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Device.DevFund.INF.TargetOSVersion

The TargetOSVersion decoration in an INF file cannot contain a ProductType flag or SuiteMask flag.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Within the [Manufacturer] section of an INF file, a TargetOSVersion decoration is used to identify the target OS of the driver package. The TargetOSVersion decoration cannot contain a ProductType flag or SuiteMask flag.

Design Notes:

In Windows 7 and earlier OS versions, the TargetOSVersion decoration is formatted as follows:

nt[Architecture].[OSMajorVersion][.[OSMinorVersion][.[ProductType][\

.[SuiteMask]]]]

Beginning in Windows 8, the ProductType field and SuiteMask field are no longer valid fields in the TargetOSVersion decoration.

Exceptions	*This is a requirement for Windows 10 Mobile, but recommended for Windows 10 for desktop editions and Windows Server Technical Preview.It will be required in the future for those architectures.

Send comments about this topic to Microsoft

Device.DevFund.Memory

Requirements related to memory profile

In this topic:

Device.DevFund.Memory.DriverFootprint

Drivers must occupy a limited memory footprint.

Applies to	Windows 10 x64 Windows 10 x86

Description

Drivers must occupy less than or equal to the following size of non-paged code pages in memory:

- Non-paged code pages

Driver Type	Graphics Drivers	All other driver types
x86/ARM	<= 10 MB	<= 1.66 MB
x64	<= 10 MB	<= 5.45 MB

- Driver locked allocations (including MDL allocations and contiguous memory allocations)
  - 12 MB for all driver types for both architectures
- Non Paged Pool - For Windows 10, drivers must occupy less than or equal to the following size of non-paged pool in memory:

Driver type	Graphics Drivers	All other driver types
x86/ARM	6 MB	4 MB

x64	10 MB	7 MB

- Thresholds are based in telemetry: X86/ARM – 4MB covers 80th percentile, X64 – 7MB covers 76th percentile from pool allocation samples

Design Notes:

The corresponding test will check the size of the drivers non-paged code pages in MB.

Additional Information

Business Justification	Driver non-paged memory usage constitutes a fixed cost in terms of memory utilization for the overall lifetime of a system. These contribute substantially toward the total OS memory footprint, and most drivers are present in memory at all times. Optimizing driver memory will provide an improved user experience and better overall system responsiveness due to greater availability of memory for user applications. Any reduction in non-pageable driver footprint directly improves the baseline memory consumption of the OS which increases scalability. Current tests for Windows 8 cover driver Locked allocations (MDL/Contiguous memory allocations) and non-paged driver code. Non Paged pool usage is the only non-pageable driver footprint aspect that is not covered by existing tests.

Device.DevFund.Memory.NXPool

All driver pool allocations must be in NX pool.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Driver pool allocations must be made in the non-executable (NX) pool.

Design Notes:

A new type of non-paged pool that is a non-executable (NX) pool has been introduced. Since it is non-executable, it is inherently more secure as compared to executable non-paged (NP) pool, and provides better protection against overflow attacks.

Additional Information

Business Justification	Moving allocations to the non-executable pool, the surface area of attack for a rogue binary's executable code is minimized.

Send comments about this topic to Microsoft

Device.DevFund.Reliability

Reliability tests containing the content of the former DEVFUND tests.

In this topic:

Device.DevFund.Reliability.BasicReliabilityAndPerformance

Drivers are architected to maximize reliability and stability and do not "leak" resources such as memory.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Driver components must not cause the system to crash or leak resources. These resources include but are not limited to the following:

- Memory
- Graphics Device Interface (GDI) or user objects
- Kernel objects such as files, mutex, semaphore, and device handles
- Critical sections
- Disk space
- Printer handles

Design Notes

To improve the reliability and stability of Windows drivers, all drivers will be subjected to a series of generic driver quality tests. These tests include:

Embedded Signature Verification Test - This test verifies that boot start drivers are embedded signed.

Device Install Check for File System Consistency - This test verifies that no system resources have been overwritten during the process of a device/driver install.

Device Install Check for Other Device Stability - This test verifies that no device or driver, except the device under test, has been affected by the device(s)/driver(s) install or co-install process.

PCI Root Port Surprise Remove Test - This test removes the PCI root port for the device (if applicable).

PNP (disable and enable) with IO Before and After - This test performs basic I/O and basic PNP disable/enable on the test device(s).

Reinstall with IO Before and After - This test uninstalls and reinstalls the drivers for test device(s) and runs I/O on these device(s).

Sleep with PNP (disable and enable) with IO Before and After - This test cycles the system through various sleep states and performs I/O and basic PNP (disable/enable) on test device(s) before and after each sleep state cycle.

Sleep with IO Before and After - This test cycles the system through various sleep states and performs I/O on device(s) before and after each sleep state cycle.

Sleep with IO During – This test cycles the system through various sleep states and performs I/O on device(s) during each sleep state cycle.

Plug and Play Driver Test - This test exercises PnP-related code paths in the driver under test.

Device Path Exerciser Test - This consists of a set of tests, each of which concentrates on a different entry point or I/O interface. These tests are designed to assess the robustness of a driver, not its functionality.

CHAOS Test - This test the runs PnP tests (disable/enable, rebalance, remove/restart, surprise remove, and DIF remove) and Driver Fuzz tests on the test device in parallel, while cycling the test system in and out of all of its supported sleep states (S1, S2, S3, S4 and Connected Standby) at the same time.

All of these tests will be run with Driver Verifier enabled with standard settings.

In addition, Driver Verifier will be enabled on all applicable kit tests.

Additional Information

Business Justification	Power management/usage, PNP errors as well as IO related errors contribute to a poor end user experience and may often cause system hangs, crashes, and failures. These tests help to identify some common driver problems.

Device.DevFund.Reliability.BasicSecurity

Device driver must properly handle various user-mode as well as kernel to kernel I/O requests (DEVFUND-0004).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Driver reliability and security are connected. Reliable drivers help protect the system from malicious attacks.

Compliance will be validated by running the Device Path Exerciser test against the device driver. Device Path Exerciser consists of a set of tests, each of which concentrates on a different entry point or I/O interface. These tests are designed to assess the robustness of a driver, not its functionality.

During a test, Device Path Exerciser typically sends hundreds of thousands of similar calls to the driver in rapid succession. The calls include varying data access methods, valid and invalid buffer lengths and addresses and permutation of the function parameters, including spaces, strings, and characters that might be misinterpreted by a flawed parsing or error-handling routine.

The device driver must comply with the reliability guidelines that are defined in the Windows Driver Kit. All user mode I/O requests and kernel-to-kernel I/O requests must be handled properly to help ensure secure devices and drivers.

Design Notes:

Potential security vulnerabilities include the failure to check for a buffer overflow, the inability to handle bad data from user mode, and the mishandling of unexpected entry points into the driver. If such vulnerabilities are left unidentified and uncorrected, malicious programs could potentially issue denial-of-service attacks or otherwise bypass system security.

For additional information, see the "Creating Reliable and Secure Drivers" and "Creating Reliable Kernel-Mode Drivers" topics in the Windows Driver Kit.

Device.DevFund.Reliability.BootDriverEmbeddedSignature

Boot drivers must be self-signed with an embedded signature.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All boot start drivers must be embedded-signed using a Software Publisher Certificate (SPC) from a commercial certificate authority. The SPC must be valid for kernel modules. Drivers must be embedded-signed through self-signing before the driver submission.

Design Notes:

For more information about how to embedded-sign a boot start driver, see Step 6: Release-Sign a Driver Image File by Using an Embedded Signature" at the following website: http://go.microsoft.com/fwlink/?LinkId=237093.

After the file is embedded-signed, use SignTool to verify the signature. Check the results to verify that the root of the SPC's certificate chain for kernel policy is "Microsoft Code Verification Root." The following command line verifies the signature on the toaster.sys file:

Signtool verify /kp /v amd64\toaster.sys

Verifying: toaster.sys

SHA1 hash of file: 2C830C20CF15FCF0AC0A4A04337736987C8ACBE3

Signing Certificate Chain:

Issued to: Microsoft Code Verification Root

Issued by: Microsoft Code Verification Root

Expires: 11/1/2025 5:54:03 AM

SHA1 hash: 8FBE4D070EF8AB1BCCAF2A9D5CCAE7282A2C66B3

Successfully verified: toaster.sys

Number of files successfully Verified: 1

Number of warnings: 0

Number of errors: 0

In the Windows Hardware Lab Kit, this requirement will be tested by using the Embedded Signature Verification test.

Additional Information

Business Justification	Boot drivers must be embedded-signed in order to work properly with the boot process.

Device.DevFund.Reliability.DriverInstallUninstallReinstall

Device and driver installation/un-installation/re-installation must be completed without any error. This includes function drivers for a multi-function device.

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Device and driver installation, uninstallation, or reinstallation must not fail in any case.

Driver installation must not cause the system to stop running or to restart without user interaction, unless the operating system requires a restart.

For multi-function devices that have separate drivers that enable separate functions, each driver must be capable of installing and uninstalling independently with no start order or hidden dependencies. A multi-function device is a single device that supports multiple functionalities.

Devices that use inbox drivers for operation must also meet this requirement. This requirement does not apply to Internet Small Computer System Interface (iSCSI) devices.

Design Notes:In the case of multi-function devices, a supervisory driver that loads different drivers for individual functions does not work well with Windows®. In particular, driver support is likely to be lost after an operating system reinstallation or upgrade, or when new drivers are distributed through Windows Update. Therefore, such supervisory drivers should be avoided.

This requirement will be tested by using the "Reinstall with IO" test.

Additional Information

Exceptions	This requirement does not apply to Internet Small Computer System Interface (iSCSI) devices. PEP and HAL extensions will be considered for exemption from this requirement.

Device.DevFund.Reliability.DriverUninstallInstallOtherDeviceStability

Installing or uninstalling the driver must not reduce or eliminate functionality of other devices or other functional parts of the same device installed on the system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Installing or uninstalling a device driver must not reduce or eliminate functionality of other devices that are installed on the system.

This requirement also applies to functional units of a multi-function device, whether that functional unit is on the multi-function device or on the system as a whole.

Design Notes:

The steps for testing this requirement are outlined in the Device install check for other device stability test: http://msdn.microsoft.com/en-us/library/ff561407(VS.85).aspx.

Device.DevFund.Reliability.NoReplacingSysComponents

Vendor-supplied drivers or software must not replace system components.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Driver or software installation must not overwrite any protected operating system files. This includes files that are not authored by Microsoft, but that are included as part of the operating system.

If a manufacturer's information file (INF) copies any files that the operating system supplies, the INF must copy those files from the Windows® product disk or preinstalled source files, unless the component is a licensed, redistributable component.

Drivers that are not provided by the operating system are not allowed to be named after an operating system supplied driver.

Device.DevFund.Reliability.NormalOpWithDEP

All drivers must operate normally and execute without errors with Data Execution Prevention (DEP) enabled.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To help ensure proper device and driver behavior and to enhance the security of Windows® systems against viruses and other security threats, all drivers must operate normally when Data Execution Prevention (DEP) is enabled. DEP monitors programs to help make sure that the programs use system memory safely. DEP also protects the system by closing any program that is trying to execute code from memory in an incorrect way.

To meet this requirement, drivers must not execute code from data pages such as default heap pages, various stack pages, and memory pool pages.

DEP is a set of hardware and software technologies that perform additional checks on memory to help prevent malicious code from running on a system. The primary benefit of DEP is to help prevent code execution from data pages. Typically, code is not executed from the default heap and the stack. Hardware-enforced DEP detects code that is running from these locations and raises an exception when execution occurs. Software-enforced DEP can help prevent malicious code from taking advantage of exception handling mechanisms in Windows.

Design Notes:

For more information about DEP, including how to enable DEP, visit the following website: http://support.microsoft.com/kb/875352.

The test for DEP is currently part of the systems test category in the Windows Hardware Certification Kit (WHCK). A device version of this test will be introduced before this requirement is enforced for logos.

Additional Information

Business Justification	DEP can help enhance the security of systems that are running Windows operating systems. DEP also helps protect against malicious code, viruses, and other security threats. Making this requirement fundamental for devices will help ensure that all drivers that are signed through the Hardware Logo Program are protected, and that the drivers prevent malware from being signed.

Device.DevFund.Reliability.PnPIDs

Plug and Play IDs embedded in hardware devices, including each functional unit of a multi-function device, must have device IDs to support Plug and Play.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Each device that is connected to an expansion bus must be able to supply its own device ID. Each function or device on a multi-function add-on device that is individually enumerated must also provide a device ID for the bus that the function or device uses.

The following are the specific requirements for Plug and Play device IDs:

- Each separate function or device on the system board must be separately enumerated. Therefore, each function or device must provide a device ID for the bus that it uses, as required in the current Plug and Play specification.
- If a device on an expansion card is enumerated, the device must have a unique ID and its own resources according to the current device ID requirements for the bus to which the card

is connected. This requirement includes devices that are separately enumerated on multi-function cards or multi-function chips.

- A Plug and Play ID can be shared with other devices that have the same model name, as defined in the device-specific Plug and Play specification.
- Each logical function of the device must have a distinct Plug and Play ID.
- The install (INF) section must key off only the most specific ID according to the Plug and Play guidelines in the Windows® Driver Kit.
- For legacy devices such as serial, parallel, and infrared ports, the legacy Plug and Play guidelines define requirements and clarifications for automatic device configuration, resource allocation, and dynamic disable capabilities.

Note: Devices that are completely invisible to the operating system, such as out-of-band systems management devices or Intelligent Input/Output (I2O) hidden devices, still must use Advanced Configuration and Power Interface (ACPI) methods to properly reserve resources and avoid potential conflicts.

The following are the exceptions to the individual device ID requirement for multi-function devices:

- Multiple devices of the same device class, such as multiline serial devices, do not need individual device IDs.
- Devices that are generated by an accelerator or auxiliary processor and that do not have independent hardware I/O do not need individual device IDs. That processor must have an ID, and the MF.sys file must be used to enumerate the dependent devices.

If an OEM uses a proprietary mechanism to assign asset or serial numbers to hardware, this information must be available to the operating system through Windows hardware instrumentation technology.

Design Notes:

See Windows Hardware Instrumentation Implementation Guidelines (WHIIG), Version1.0, at the following website:

http://go.microsoft.com/fwlink/?LinkId=237095.

Additional Information

Business Justification	A unique Plug and Play ID provides a good end user experience for devices. Because a unique device installs each device driver, this requirement helps prevent the issues that occur in Windows Update. This requirement now also includes all aspects of Plug and Play that are relevant for multi-function devices to enable a good Plug and Play experience when the device is used with Windows. This requirement

	enhances compatibility and reliability when Windows is used with certified devices.

Device.DevFund.Reliability.PnPIRPs

Drivers must support all PnP IRPs

Applies to	Windows Server 2016 Technical Preview x64

Description

Drivers must support all Plug and Play I/O request packets (IRPs) according to the requirements on the following website:

http://msdn.microsoft.com/en-us/library/ff558807(v=VS.85).aspx

The following IRPs are often the cause of driver issues. Special attention should be given to their implementation:

- Removal
  - IRP_MN_QUERY_REMOVE_DEVICE
  - IRP_MN_CANCEL_REMOVE_DEVICE
  - IRP_MN_REMOVE_DEVICE
- Rebalancing
  - IRP_MN_QUERY_STOP_DEVICE
  - IRP_MN_QUERY_RESOURCE_REQUIREMENTS
  - IRP_MN_FILTER_RESOURCE_REQUIREMENTS
  - IRP_MN_CANCEL_STOP_DEVICE
  - IRP_MN_STOP_DEVICE
  - IRP_MN_START_DEVICE
  - IRP_MN_REMOVE
- Surprise Removal
  - IRP_MN_SURPRISE_REMOVAL

Device.DevFund.Reliability.ProperINF

Device driver must have a properly formatted INF for its device class (DEVFUND-0001).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Driver installation and removal must use Windows®-based methods. Therefore, only information file (INF)-based installation routines are allowed. A device driver must have a properly formatted INF for its device as described in the Windows Driver Kit (WDK) "Creating an INF File" topic.

Design Notes:

The "INFTest against a single INF" test validates this requirement. For more information about this test, see the Help documentation of the test kit.

Note: If the device does not provide an INF file (that is, the device uses the inbox driver and the INF file only), this requirement does not apply.

Additional Information

Exceptions	If the device does not provide an INF file (that is, the device uses the inbox driver and the INF file only), this requirement does not apply.

Device.DevFund.Reliability.RemoteDesktopServices

Client and server devices must function properly before, during, and after fast user switching or a Microsoft Remote Desktop Services session (DEVFUND-0009).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Devices must support Fast User Switching (FUS) and Remote Desktop Services without losing data before, during, or after sessions. Any user interface (UI) for the device must be shown in the session to which the UI applies. Device usage must not be indefinitely blocked in alternate user sessions.

Additional Information

Business Justification	FUS and Remote Desktop Services are Windows® features. To provide a good and consistent user experience, each device needs to work properly with these services.

Device.DevFund.Reliability.PCSupportsLowPowerStates

All devices and drivers must support S4 and S5 and either S0 low power idle or S3 sleep states of the system they are integrated on or connected to.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All devices and drivers must meet the following requirements for systems that are entering S4 (Hibernate) and S5 (Soft-off) and either S0 low power idle, or S3 (Sleep):

- All devices and drivers must correctly support the request of a system that is going into S0, S3, S4, or S5 states.
- Devices and drivers must not veto the request from the system.
- The devices must support both the S0, S3, S4, and S5 states.
- All devices must be capable of resuming from S0, S3, S4, and S5 sleep states and be fully functional after waking up.
- The device and all its functional units (in the case of multi-function devices) must be enumerated appropriately after the device resumes.
- All devices and drivers must respond properly to Plug and Play events, IOCtl calls, and I/O requests that are issued concurrently with sleep state transitions.

This requirement helps to ensure that all certified and signed devices will support the S0, S3, S4, and S5 sleep states when the devices are used as part of a system or are connected externally to a system. This requirement will help the systems to conserve power when the system is not being used. Power management is an important aspect of a good user experience. The system should be able to control which devices are put into a sleep state when the devices are not being used. All devices must comply with the request from the system to go into a sleep state and not veto the request, thereby putting an additional drain on the power source.

Design Notes:

This requirement will be tested by using the "Sleep Stress with IO" test and the "PnPDTest with Concurrent IO in parallel with DevPathExer" test in the Windows Hardware Lab Kit (HLK).

The system that is used for testing must support S0, S3, S4, and S5.

Note that systems that support Connected Standby will not support S3, and may or may not support S4. Devices in such systems must support Connected Standby, and S4 requests of that system (if applicable).

Device.DevFund.Reliability.Signable

Device drivers must be able to be signed by Microsoft.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All devices must have code signed drivers. Drivers that are submitted for the Windows® certification must be able to be code signed and must meet the Microsoft® guidelines that are defined in the Windows Driver Kit "WHQL Digital Signatures" topic. All boot start drivers must be embedded-signed by using a certificate that is valid for kernel modules. Devices must be embedded-signed via self-signing before the devices are submitted to Microsoft for certification. It is recommend that all drivers are embedded-signed via self-signing before the drivers are submitted to Microsoft, although this is only required for boot start drivers.

Design Notes:

For requirements for digital signatures, see the "Driver Signing/File Protection" topic at the following website:

http://go.microsoft.com/fwlink/?LinkId=36678.

The INF2CAT signability verification tool installs automatically the first time that you create a submission on Microsoft's website. For more information about the INF2CAT tool, visit the following website:

http://go.microsoft.com/fwlink/?LinkId=109929.

Additional Information

Exceptions	This requirement does not apply to devices that use the inbox drivers of the operating system.

Device.DevFund.Reliability.SWDeviceInstallsUsePnPAPIs

Software-initiated device-installs must use Plug and Play APIs to install device drivers.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Device installers that directly manipulate Plug and Play resources contribute to system instability. Therefore, direct manipulation of Plug and Play resources will be blocked on later releases of Windows®. To help to ensure compatibility with Windows releases, standard Plug and Play application programming interfaces (APIs) must be used to install device drivers.

Design Notes:

In Windows Vista® and later operating systems, standard Plug and Play calls such as the **SetupCopyOEMInf** call pre-stage all required files for device installation on the system automatically. Pre-staging of driver packages will facilitate driver package migration during a system upgrade to Windows Vista or later Windows operating systems. We strongly encourage the use of the Driver Install Framework tools to meet this logo requirement. The use of DIFxAPI, DIFxAPP, or DPInst DIFx tools fulfills this requirement.

Send comments about this topic to Microsoft

Device.DevFund.Reliability.3rdParty

Reliability tests containing content of the former DEVFUND tests.

In this topic:

Device.DevFund.Reliability.3rdParty.FormerTests

Former Tests Mapping Requirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The feature Device.DevFund.Reliability.3rdParty and this requirement are a placeholder for mapping of former DevFund tests that are not found in the other requirements.

Additional Information

Exceptions	This requirement does not apply to devices that use the inbox drivers of the operating system.

Send comments about this topic to Microsoft

Device.DevFund.Reliability.Interrupts

Reliability with respect to device interrupts

In this topic:

Device.DevFund.Reliability.Interrupts.BasicReliabilityAndPerformance

Drivers must not exceed the maximum number of interrupts, and must support resource arbitration down to a minimum level as defined by the operating system

Applies to	Windows Server 2016 Technical Preview x64

Description

The driver must be able to tolerate system re-balancing of interrupt resources with any alternative chosen by the OS without failures, including the theoretical minimum of one line based interrupt.

Interrupt arbitration may require multiple iterations. Drivers must be prepared to tolerate cases where their initial interrupt request is rejected. In order to support optimal and timely interrupt arbitration, drivers should provide multiple alternatives at successively reduced interrupt count. Drivers should avoid requesting more than one interrupt per core when possible. Any request for greater than 2048 interrupts per device function will be rejected per the PCIe 3.0 defined MSI-X table entry limit of 2048 per device.

Additional Information

Business Justification	Requesting more than one interrupt per core can lead to IDT exhaustion in settings where many devices are present. Requesting a total number of interrupts based on the number of cores often leads to memory allocation issues.

Send comments about this topic to Microsoft

Device.DevFund.ReliabilityDisk

Reliability tests targeting disk devices

In this topic:

Device.DevFund.ReliabilityDisk.IOCompletionCancellation

A device driver must follow the design details in the I/O Completion/Cancellation Guidelines (DEVFUND-0013).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

I/O completion and cancellation guidelines for drivers provide prescriptive requirements to help ensure that device drivers do not stop responding and can be fully cancelled. These requirements also suggest best practices that drivers can follow to improve performance.

Based on the guideline, all device drivers must meet the following requirements:

- All I/O request packets (IRPs) that are cancelled must be completed within five seconds in an environment that is not resource constrained. No cancellation should be missed even though the cancellation requests may arrive at any instant, even before a driver's dispatch routine sees the IRP.
- All resources that a cancelled IRP allocates must be released at IRP cancellation time to prevent hindering system performance under a high cancellation load. The cancellation of the IRP should shut down any I/O intensive process.

In addition, we strongly recommend that drivers do not depend on an additional allocation of resources during cancellation.

Design Notes:

The Windows® I/O Manager includes enhancements to support cancellation of the MJ_IRP_CREATE process. The Win32 application programming interfaces (APIs) include enhancements to support the cancellation of synchronous operations, such as CreateFile. These enhancements allow I/O cancellation in more environments than in earlier operating systems. For more information, see the "I/O Completion/Cancellation Guidelines" whitepaper at the following website: http://www.microsoft.com/whdc/driver/kernel/default.mspx.

For more information about designing completion and cancellation logic for drivers, see the following topics in the Windows Development Kit (WDK):

Completing IRPs

Canceling IRPs

Cancel-Safe IRP Queues

Using the System's Cancel Spin Lock

Additional Information

Business Justification	The primary justification for this requirement is to prevent drivers from causing applications to stop responding. Additionally, users cannot terminate or restart the applications. Drivers that cause applications to stop responding are a significant cause of customer dissatisfaction. A secondary justification for this requirement is to improve the customer experience by permitting I/O cancellation to occur on demand by a user, through the use of user interface (UI) elements such as a universal stop button or cancel buttons. This requirement was introduced in Windows Vista. The requirement adds demands to existing driver cancellation logic and adds the requirement that drivers support cancelling creation requests. Drivers that stop responding can lead to sometimes random application and operating system failures that result in lost customer productivity, lost customer data, and the need to reboot the computer. Beyond these very serious problems, nonexistent or poor I/O cancellation support prevents applications from successfully stopping operations without restarting the application.

Send comments about this topic to Microsoft

Device.DevFund.Security

Additional TDI filter driver and LSP requirements related to security.

In this topic:

Device.DevFund.Security.NoTDIFilterAndLSP

No TDI filters or LSPs are installed by the driver or associated software packages during installation or usage.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

There can be no use of TDI filters or LSPs by either kernel mode software or drivers, or user mode software or drivers.

Additional Information

Business Justification	Use of TDI filters and LSPs increase attack surface, and will therefore no longer be supported for future OS releases.

Send comments about this topic to Microsoft

Device.DevFund.Server

In this topic:

Device.DevFund.Server.CommandLineConfigurable

Windows Server device drivers which have configurable settings provide command line utility or function for device and driver management

Applies to	Windows Server 2016 Technical Preview x64

Description

Windows® Server® device drivers that are configurable are required to supply a WMI or PowerShell-based, remoteable functionality for device and driver management. The following device categories are included:

- Network, including teaming and Infiniband
- Storage, including multipath I/O (MPIO)
- Bus
- Any other drivers that may have configurable settings

The specific requirements are the following:

- The utility or tool functionality may use either PowerShell or Windows Management Instrumentation (WMI) functionality that the Windows Nano Server installation option supports.
- The utility or functionality must operate from the command line or be a WMI object and provider that is compatible with the Windows Management Instrumentation Command-line (WMIC) tool.
- The utility must be provided as part of the driver package and be installed by default on the system with the driver.
- The utility must be able to correctly query, display, and change any values that can be changed for the driver.
- The utility must not incorrectly create or set options that do not exist for a specific device or driver. The utility must be capable of changing any setting if the operating system does not provide the ability to change that setting from the command line.
- Changed values or ranges that the user inputs must be automatically checked to ensure that the user input is valid.
- Changes that the utility makes must not require any network or storage stack restarts or system reboots, unless a boot, system, or paging behavior or location is modified.
- Changes that the utility makes are persistent across reboots.
- Help about the utility usage and options is available locally on the system. For example, the utility must provide a "/?" command-line option, or the WMI options for the product must be exposed through standard WMIC commands.
- The utility should not be installed by the information file (INF).
- The utility should be installed by default. This can be accomplished by using a co-installer.

This requirement does not apply to storage arrays, storage fabrics, switches, printers, or other devices that are external to the server system and that can be managed by any system that is attached to the Ethernet, Fibre Channel, or other network.

This requirement does not apply to any device that does not have configurable settings. For example, in a system that uses the graphical interface, there are no "Advanced", "Power Management", or other additional tabs in the Device Manager interface, nor are any utilities available from the vendor that achieve the same effect.

This requirement applies to any physical device, or device that has a PCI ID, that has a driver; to any driver that is in the network, storage, file system or other stacks; and to any device or driver that otherwise operates at kernel or user mode in the operating system instance on the server.

Design Notes:

Any device driver that does not meet this requirement will not be usable on Nano Server systems

Enforcement Date: Feb 2016

Device.DevFund.Server.MultipleProcessorGroups

Drivers must operate correctly on servers that have processors configured into multiple processor groups

Applies to	Windows Server 2016 Technical Preview x64

Description

Windows® Server uses the concept of KGROUPs to extend the number of processors that a single instance of the operating system can support to more than 64. Both enlightened and unenlightened device drivers must operate correctly in multiple KGROUP configurations.

Design Notes:

For more information, see the "Supporting Systems That Have More Than 64 Processors: Guidelines for

Developers" white paper at the following website:

http://www.microsoft.com/whdc/system/Sysinternals/MoreThan64proc.mspx

This requirement is tested for all server device categories. The test uses BCDEdit functionality to change the boot configuration database (BCD) of the operating system, thus changing the size of the processor groups (the

groupsize setting) so that multiple processor groups are created. The test also uses BCDEdit functionality to add the **groupaware** setting to the BCD. This changes the behavior of several now-legacy application programming interfaces (APIs) so that the test finds more code errors.

The operating system will not ship with any of these settings. These settings are for testing only and will not be supported in production. To reconfigure the system for normal operations, these settings must be removed from the BCD and the system must be rebooted. The system that is used for testing must include at least four processor cores.

Vendors may configure the system so that it is similar to the HLK and Device Test Manager

(DTM) systems. Vendors can perform their own tests in a multiple processor group configuration, as follows.

The command lines to add the group settings and reboot the computer are the following:

```
bcdedit.exe /set groupsize 2
bcdedit.exe /set groupaware on
```

```
shutdown.exe -r -t 0 -f
The command lines to remove the group settings and reboot the computer
are the following:
bcdedit.exe /deletevalue groupsize
bcdedit.exe /deletevalue groupaware
shutdown.exe -r -t 0 -f
```

Device.DevFund.Server.OperateInServerCore

Device drivers must install, configure, be serviced, and operate in Windows Nano Server and Server Core.

Applies to	Windows Server 2016 Technical Preview x64

Description

The hardware platforms on which Windows Server operating systems are deployed have evolved dramatically in the past decade. As these become graphic-less system designs for cost and deployment efficiencies, the customers expect to completely setup, deploy, configure and manage these hardware platforms using the minimal command line interface and automated scripting of Windows Nano Server and Server Core. Windows Server device drivers must evolve in a similar manner to allow the customers to pursue these operations unhindered.

A device driver must demonstrate its ability to install, configure, be serviced and operate without reliance on the presence of a GUI.

Design Notes:

Any device driver that does not meet this requirement will not be usable on Nano Server systems

Enforcement Date: Feb 2016

Device.DevFund.Server.ServerPowerManagement

Windows Server device drivers must support Query Power and Set Power Management requests

Applies to	Windows Server 2016 Technical Preview x64

Description

During transition into the hibernation (S4) system sleep state, device drivers receive power requests from the operating system. Device drivers must honor and must correctly handle the Query Power and Set Power power management requests. A device driver should never reject a Query Power request. When a device driver receives an S4 Set Power power management request, the driver must transition its devices into a D3 device power state and stop all I/O operations. This includes any direct memory access (DMA) transfers that are currently in progress. When the driver's devices are in a low power state, interrupts are disabled, and all in-progress I/O operations are halted.

The D3 state may be either D3 "Hot", to support devices that must respond to external stimuli, or D3 "Cold".

A device driver must not bind itself to a uniquely identifiable instance of system hardware, such as a specific processor.

Design Notes:

For more information, see the "Driver Compatibility for Dynamic Hardware Partitioning" white paper at the following website: http://www.microsoft.com/whdc/system/platform/server/dhp.mspx

Send comments about this topic to Microsoft

Device.DevFund.Server.PCI

PCI

In this topic:

Device.DevFund.Server.PCI.PCIAER

Windows Server PCI Express devices are required to support Advanced Error Reporting [AER] as defined in PCI

Express Base Specification version 2.1.

Applies to	Windows Server 2016 Technical Preview x64

Description

See Tables 6-2, 6-3, 6-4 and 6-5 of the PCI Specification on how errors are detected in hardware, the default severity of the error, and the expected action taken by the agent which detects the error with regards to error reporting and logging.

All three methods of error reporting; completion status, error messages, error forwarding\data poisoning.

Completion status enables the Requester to associate an error with a specific Request.

Error messages indicate if the problem is correctable or not, and fatal or not

Error forwarding\data poisoning can help determine if a particular Switch along the path poisoned the TLP

The following table lists which errors in section 6.2 are required to be reported:

Type of Errors	Required?	Action
ERR_COR (correctable)	No	No action, not logged in Event Viewer, system takes no action.

ERR_FATAL (fatal, non-correctable)	Yes	WHEA handles, and logged in Event Viewer
ERR_NONFATAL	Yes	None

Send comments about this topic to Microsoft

Device.DevFund.Server.StaticTools

In this topic:

Device.DevFund.Server.StaticTools.SDVandPFD

Driver development includes static analysis to improve reliability using Static Driver Verifier (SDV) and Prefast for Drivers (PFD).

Applies to	Windows Server 2016 Technical Preview x64

Description

Server driver development must include passing log files for Static Driver Verifier (SDV) and Code Analysis (CA).

Design Notes:

Microsoft's Static Analysis Tools, namely, Code Analysis (CA) and Static Driver Verifier (SDV) have been found to be highly effective in improving driver reliability by identifying coding issues that would be otherwise difficult to find. Errors or warnings output by CA and SDV that are determined to be “Server: Must Fix” must be corrected in driver code of devices submitted for Server Network and Storage certification.

The results file will be captured by the Windows Hardware Lab Kit for inclusion in the submission package.

Note that there is no requirement that the submitted and subsequently distributed binary be compiled using Microsoft Windows Device Kit or other tools.

Send comments about this topic to Microsoft

Device.Display.Monitor

In this topic:

Device.Display.Monitor.Base

Base requirements for displays must ensure a good end user experience.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All connectors on the monitor must be set to a mode that will not apply CE style overscan or underscan by default. It is ok for the monitor to provide an option to allow the user to configure overscan/underscan using an on screen display.

All video displays that provide an HDMI connector, must support the ITC flag as defined in the HDMI specification.

All digital displays are required to have a single HPD signal transition from low to high on device connection and power up. Periodic toggling of the HPD signal after connection or power up is not allowed.

Multiple transition lead source to notify the OS of multiple device arrival and removal event; causing undesirable mode set flashing.

Device.Display.Monitor.DigitalLinkProtection

Display monitors that support digital inputs must support digital link protection on all digital inputs.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Displays with digital inputs, such as Digital Visual Interface (DVI), High-Definition Multimedia Interface, (HDMI), DisplayPort, etc.. must support a digital monitor link protection mechanism such as High-bandwidth Digital Content Protection (HDCP).

Device.Display.Monitor.EDID

A display device must implement the EDID data structure.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The monitor must transmit an EDID structure that contains all required fields, as defined in VESA Enhanced Extended Display Identification Data Standard (E-EDID), Release A, Section 3. This EDID must also contain a unique Manufacturer Name, Product code ID, and Serial Number. (The serial number is not required for an integrated panel on a mobile or all in one system.)

For analog CRTs, EDID content must indicate at least one VESA mode at 75 Hz or higher for each supported resolution.

All monitors must support E-EDID by implementing an EDID 1.3 or later data structure that:

- Includes timing data for the preferred display mode in Timing #1.
    - For an LCD or other fixed-format display, this display mode is the native, progressively scanned mode of the panel.
    - For other display types, this is the optimal, progressively scanned display mode that is based on the size and capabilities of the device, and must meet the requirements for refresh rates defined above.
- Implements the screen size or aspect ratio fields, bytes 0x15 and 0x16 per the supported EDID version with accurate dimensions.
- Sets byte 0x18, Bit 1 to indicate that the preferred mode meaning per the supported EDID version.
- Includes a unique serial number in at least one of the ID Serial Number field or a Display Product Serial Number string in one of the base block 18-byte descriptors.
- Implements a Display Product Name string in one of the base block 18-byte descriptors, optional for an integrated panel. This string must be suitable for user interface usage.

- Implements a Display Range Limits in one of the base block 18-byte descriptors, unless the device is a Non-Continuous Frequency (multi-mode) display.

Mobile and other all-in-one systems must transmit an EDID structure in one of three ways:

- LCD panel provides one, which is similar to an externally attached monitor.
- If the LCD panel does not provide one, then the WDDM miniport is responsible for defining and providing it to the operating system.
- The WDDM driver may execute the ACPI _DDC method on the child device associated with the internal panel to retrieve an EDID from the system BIOS.

Display devices that implement features that have more than 8 bits per primary color must use EDID 1.4 in order to ensure that these capabilities can be expressed to the OS and applications.

Design Notes:

The ACPI specification defines the method to acquire the EDID from the BIOS to achieve equivalent functionality as specified in ACPI 2.0b, Appendix B, or later.

Device.Display.Monitor.Modes

Requirement for resolution support for display devices

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A display device can have multiple connectors. The following are the required modes that a display must support on each connector and indicate the support via the EDID (a display is free to support additional modes and call them out in the EDID as well).

For an integrated panel:

The native resolution of the panel must be greater than or equal to 1024 x 768. The native resolution must be supported at 60 Hz progressive or greater or the closest frequency appropriate for the region.

For HD15, DVI, HDMI, and DisplayPort connector:

The native resolution of the panel must be greater than or equal to 1024 x 768. The native resolution must be supported at 60 Hz progressive or greater or the closest frequency appropriate for the region.

The following modes must be supported by the display and included in the Established timings in the EDID:

- 640 x 480 at 60 Hz progressive (Byte 23h, bit 5 in the Established timing)
- 800 x 600 at 60 Hz progressive (Byte 23h, bit 0 in the Established timing)
- 1024 x 768 at 60 Hz progressive (Byte 24h, bit 3 in the Established timing)

These modes can be supported as full screen or centered.

For all other connectors like S-Video, Component, and Composite:

The connector must support the maximum allowable mode as defined in the specification of the standard.

Device.Display.Monitor.Stereoscopic3DModes

A stereo 3D external display or internal mobile panel must support a stereo mode equivalent to its native or preferred resolution.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The native or preferred resolution of the stereo 3D display must have an equivalent stereo mode. The native or preferred resolution of the display is exposed through its EDID.

Example: If the native resolution of the stereo 3D display is 1920 x 1200 in mono, then it must also support the same native resolution in the stereo mode.

Send comments about this topic to Microsoft

Device.Graphics.AdapterBase

The base feature set required by all graphic devices.

In this topic:

Device.Graphics.AdapterBase.ApplicationVerifier

Graphics driver components must comply with the Application Verifier test criteria.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All user-mode modules (.dll or .exe files) that are part of a graphics driver must satisfy the test criteria for Application Verifier tests. During the testing of the driver, Application Verifier must be turned on for processes where the driver modules are executing. Application Verifier will cause a break when an error is detected.

For graphics drivers, AppVerifier is required for critical executables (i.e. DWM.exe as an example) used to test the stability or robustness of the graphics driver.

In addition, Application Verifier must be enabled on IHV's control panel executable as part of this requirement.

These Application Verifier tests must be turned on for the processes during testing:

- com
- exceptions
- handles
- heaps
- inputoutput
- leak
- locks
- memory
- rpc
- threadpool
- tls
- hangs

Device.Graphics.AdapterBase.DriverVersion

The driver DLL for a graphics adapter or chipset has a properly formatted file version.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The file version of the graphic driver DLLs (UMD, KMD) and .SYS files must match each other and must be of the form:

A.BB.CC.DDDD

- The A field must be set to 20 for WDDM 2.0 drivers on Windows 10.
- The A field must be set to 10 for WDDM 1.3 drivers on Windows 8.1.
- The A field must be set to 9 for WDDM 1.2 drivers on Windows 8.
- The A field must be set to 8 for WDDM 1.1 drivers on Windows 7.
- The A field must be set to 7 for WDDM 1.0 drivers on Windows Vista.
- The A field must be set to 6 for XDDM drivers on Windows Vista.

For Windows 7 and earlier (WDDM 1.1 and earlier) drivers, the BB field must be set to the DDI version that the driver supports:

- DirectX 9 drivers (which expose any of the D3DDEVCAPS2_* caps) must set BB equal to 14.
- DirectX 10 drivers must set BB equal to 15.
- D3D11-DDI driver on D3D10 hardware must set BB equal to 16.
- D3D11-DDI driver on D3D11 hardware must set BB equal to 17.

For Windows 8 (WDDM 1.2) drivers, the BB field must be set to the highest DirectX Feature Level supported by the driver on the graphics hardware covered by the driver:

- A Feature Level 9 driver must set BB equal to 14.
- A Feature Level 10 driver must set BB equal to 15.
- A Feature Level 11 driver must set BB equal to 17.
- A Feature Level 11_1 driver must set BB equal to 18.
- A Feature Level 12 driver must set BB equal to 19.

The CC field can be equal to any value between 01 and 99.

The DDDD field can be set to any numerical value between 0 and 9999.

For example:

Windows Vista DirectX 9.0-compatible WDDM drivers can use the range 7.14.01.0000 to 7.14.99.9999.

Windows 7 DirectX 10.0-compatible WDDM 1.1 drivers can use the range 8.15.01.0000 to 8.15.99.9999.

Windows 8 WDDM 1.2 driver on DX10 hardware would be 9.15.99.9999.

Device.Graphics.AdapterBase.PowerManagementCompliance

A graphics adapter must comply with VESA and ACPI power management specifications to enable system sleep.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To ensure correct implementation of operating system-controlled power management, graphic adapters and their drivers must respond to:

- Required device (D0 and D3) power states as highlighted in the WDK
- Operating system power management for ACPI power states
- *VESA BIOS Power Management Functions, which defines extensions to VGA ROM BIOS services for power management. (*This line does not apply to UEFI GOP based platforms.)

Device.Graphics.AdapterBase.RegistryEntries

An application that installs the device driver for a graphics device must create the required registry entries.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following registry entries must be created during video driver installation:

- REG_BINARY: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video\PART_GUID\ID\InstalledDisplayDrivers: This key should contain the User-mode driver name.
- REG_BINARY: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video\PART_GUID\ID\HardwareInformation.MemorySize

The below Hardware Information values are written by the WDDM driver:

- REG_BINARY: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video\PART_GUID\ID\HardwareInformation.ChipType
- REG_BINARY: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video\PART_GUID\ID\HardwareInformation.DACType
- REG_BINARY: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video\PART_GUID\ID\HardwareInformation.AdapterString
- REG_BINARY: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video\PART_GUID\ID\HardwareInformation.BiosString
- REG_BINARY: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Video\PART_GUID\ID\HardwareInformation.MemorySize

The following INF directives must be included in the INF:

- Feature Score - This is a General Installation setting that must be supported by all WDDM drivers.
- Copy Flags to Support PNP Stop directive
- Driver\Services Start Type directive
- Capability Override settings to disable OpenGL
- [Version] section directives
- [SourceDiskNames] section directives

- General x64 directives
- General [Install] section directives

The Driver DLL must have a properly formatted file version as defined in the requirement Device.Graphics.AdapterBase.DriverVersion.

Device.Graphics.AdapterBase.SubsystemResettable

A graphics subsystem must be resettable to a normal operational state.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the GPU is hung for any reason, independent of what the hardware is processing at the time, it must be resettable to a normal operational state. This basically implies that TDR must be supported by any GPU.

Hybrid system should be able to handle TDR just like non-hybrid system and have mechanism to reset either (or both) the GPU to bring the system back to a functioning state when the operating system detects a hang.

Design Notes:

The ability to reset the GPU is independent of the current working state of the system.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender

The Render feature of a graphics device.

In this topic:

Device.Graphics.AdapterRender.MinimumDirectXLevel

A graphics adapter must implement the minimum hardware acceleration capabilities.

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

The display subsystem is required to implement the DirectX 9 hardware specification and the driver is required to expose through the D3D9 UMD DDI the capabilities of Direct3D 10 Feature Level 9_3 as described in MSDN here:

- http://msdn.microsoft.com/en-us/library/ff476876(v=VS.85).aspx
- http://msdn.microsoft.com/EN-US/library/ff476150.aspx
- http://msdn.microsoft.com/EN-US/library/ff476149.aspx
- http://msdn.microsoft.com/en-us/library/ff471324(v=VS.85).aspx

Device.Graphics.AdapterRender.RGBFrameBuffer

A display chipset must implement the specified component order and endian representation for an 8-bpp or greater integer RGB frame buffer formats.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

For an N-bit-per-component RGB frame buffer format, the lowest N bits must contain the blue component, the next N bits must contain the green component, the next N bits must contain the red component, and the remaining 32-(3 x N) bits may contain alpha. The resulting 32-bit value must be stored in memory in little-endian format.

Device.Graphics.AdapterRender.YUVSupport

A display driver that supports YUV textures must process textures and functions correctly.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

If the hardware supports YUV texture surfaces and the capability is reported, then the driver must be able to process these surfaces without any intermediate transforms and function correctly.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D101Core

D3D 10.1 core feature

In this topic:

Device.Graphics.AdapterRender.D3D101Core.D3D101CorePrimary

If the graphics device supports Direct3D 10.1, it must comply with the Direct3D 10.1 and DXGI Specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the graphics devices implements Direct3D 10.1, it must meet all the requirements defined in the Direct3D 10.1 specification and must provide sufficient performance for Direct3D 10.1 features.

Since Direct3D 10.1 is a superset of Direct3D 10, the implementation of Direct3D 10.1 also requires the support of the Direct3D 10 feature set.

All features required by this specification must be exposed by the device driver including those features defined for the DXGI DDI header.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D101WDDM11

D3D 10.1 core feature with WDDM 1.1 additions

In this topic:

Device.Graphics.AdapterRender.D3D101WDDM11.D3D101v11Primary

If a WDDM 1.1 graphics device supports Direct3D 10.1, it must comply with the Direct3D 10.1 and DXGI Specifications and support BGRA.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the graphics hardware implements Direct3D 10.1, it must meet all the requirements defined in the Direct3D 10.1 specification and must provide sufficient performance for Direct3D 10.1 features.

Since Direct3D 10.1 is a superset of Direct3D 10, implementation of Direct3D 10.1 also requires support of the Direct3D 10 feature set. Additionally, the following features originally defined in the D3D9 hardware specification are now required:

- BGRA

All features required by this specification must be exposed by the device driver including those features defined for the DXGI DDI header.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D101WDDM12

D3D 10.1 core feature with WDDM 1.2 additions

In this topic:

Device.Graphics.AdapterRender.D3D101WDDM12.D3D101v12Primary

If a WDDM 1.2 graphics device supports Direct3D 10.1, it must comply with the Direct3D 10.1, DXGI and D3D10 portion of the Direct3D 11.1 Feature Specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the graphics hardware implements Direct3D 10.1, it must meet all requirements that are defined in the Direct3D 10.1 specification and must provide sufficient performance for Direct3D 10.1 features.

Since Direct3D 10.1 is a superset of Direct3D 10, the implementation of Direct3D 10.1 also requires support of the Direct3D 10 feature set. Additionally, the following features originally defined in the D3D9 hardware specification are now required:

- BGRA
- Half Precision pixel formats (5551, 565, 4444)
- Same Surface Blts

Please see the Direct3D 11.1 Features Spec for complete details of all new features required to be exposed with a WDDM 1.2 driver for D3D10+ hardware.

All features required by this specification must be exposed by the device driver, including those features defined for the DXGI DDI header.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D10ComputeShader

D3D10* Shader Model 4_* Compute Shader Functionality

In this topic:

Device.Graphics.AdapterRender.D3D10ComputeShader.D3D10CoreC

If the graphics hardware implements D3D10* Shader Model 4_* Compute Shader Functionality, it must conform to the D3D11 hardware specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D11 Specification allows for optionally implementing Shader Model 4_* Compute Shader functionality on D3D10* hardware. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D11 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D10Core

D3D 10 core feature

In this topic:

Device.Graphics.AdapterRender.D3D10Core.D3D10CorePrimary

If a graphics device supports Direct3D 10, it must comply with the Direct3D 10 and DXGI specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the graphics hardware implements Direct3D 10, it must meet all requirements defined in the Direct3D 10 specification and must provide sufficient performance for Direct3D 10 features.

All features required by this specification must be exposed by the device driver, including those features defined for the DXGI DDI header. The following list includes some of the required features called out in the Direct3D 10 specification:

- Geometry shader
- Stream output
- Integer instruction set
- New compressed formats
- Render to vertex buffer
- Render to cube map
- Render to volume

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D10D3D11LogicOps

D3D10-D3D11 Logic Ops

In this topic:

Device.Graphics.AdapterRender.D3D10D3D11LogicOps.D3D10CoreD

If the graphics hardware implements Logic Ops functionality, it must conform to the D3D11 hardware specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D11.1 Specification allows for optionally implementing Logic Ops functionality on D3D10, D3D10.1 and D3D11 hardware. If the hardware supports and exposes support for this functionality, it must conform to the specifications for this feature as defined in the D3D11.1 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D10Multisampling4X

D3D10 Multisampling (4X)

In this topic:

Device.Graphics.AdapterRender.D3D10Multisampling4X.D3D10CoreA

If the graphics hardware implements D3D10 4x Multisampling, it must conform to the D3D10 hardware specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D10 Specification allows for optionally implementing 4X Multisampling. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D10 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D10Multisampling8X

D3D10* Multisampling (8X)

In this topic:

Device.Graphics.AdapterRender.D3D10Multisampling8X.D3D10CoreB

If the graphics hardware implements D3D10* 8X Multisampling, then it must conform to the D3D10 hardware specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D10 Specification allows for optionally implementing 8X Multisampling. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D10 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D10WDDM11

D3D 10 core feature with WDDM 1.1 additions

In this topic:

Device.Graphics.AdapterRender.D3D10WDDM11.D3D10v11Primary

If the graphics hardware implements Direct3D 10 and the driver is WDDM1.1, it must comply with the Direct3D 10 and DXGI Specifications and support BGRA.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the graphic hardware implements Direct3D 10 and the driver is WDDM1.1, it must meet all requirements defined in the Direct3D 10 specification and must provide sufficient performance for Direct3D 10 features. Additionally, the following features originally defined in the D3D9 Hardware specification are now required:

- BGRA

All features required by this specification must be exposed by the device driver including those features defined for the DXGI DDI header. The following list includes some of the required features called out in the Direct3D 10 specification:

- Geometry shader
- Stream output
- Integer instruction set
- New compressed formats
- Render to vertex buffer
- Render to cube map
- Render to volume

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D10WDDM12

D3D 10 core feature with WDDM 1.2 additions

In this topic:

Device.Graphics.AdapterRender.D3D10WDDM12.D3D10v12Primary

If the graphics hardware implements Direct3D 10 and the Driver is WDDM1.2, it must comply with the Direct3D 10, DXGI, and the D3D10 additions in the Direct3D 11.1 Feature Specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the graphic hardware implements Direct3D 10 and the driver is WDDM1.2, it must meet all requirements defined in the Direct3D 10 specification and must provide sufficient performance for Direct3D 10 features. Additionally, the following features originally defined in the D3D9 Hardware specification are now required:

- BGRA
- Half Precision pixel formats (5551, 565, 4444)
- Same Surface Blts

Please see the Direct3D 11.1 Features Spec for complete details of all new features required to be exposed with a WDDM 1.2 driver for D3D10+ hardware.

All features required by this specification must be exposed by the device driver including those features defined for the DXGI DDI header. The following list includes some of the required features called out in the Direct3D 10 specification:

- Geometry shader
- Stream output
- Integer instruction set
- New compressed formats
- Render to vertex buffer
- Render to cube map
- Render to volume

Device.Graphics.AdapterRender.D3D10WDDM12.Stereoscopic3DArraySupport

WDDM1.2 drivers must support Stereoscopic 3D in D3D by adding expanded array support.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Support required by WDDM 1.2

DDI or feature	**WDDM 1.1**	**WDDM 1.2**
Creation of backbuffers (and primaries) using D3D10DDIARG_CREATERESOURCE & D3D11DDIARG_CREATERESOURCE	Restricted to ArraySize = 1	Generalized to ArraySize = 2
DXGI UM backbuffer DDIs (DXGI_DDI_ARG_BLT, DXGI_DDI_ARG_ROTATE_RESOURCE_IDENTITIES, DXGI_DDI_ARG_PRESENT, DXGI_DDI_ARG_SETDISPLAYMODE)	Restricted to backbuffers	Same, but including stereo back buffers
Presentation in general (kernel and user)	Restricted to backbuffers	Same, but including stereo back buffers
Sharing	Restricted to ArraySize = 1	Generalized to any ArraySize (including > 2)
GDI interop	Restricted to ArraySize = 1	Generalized to any ArraySize (including > 2)
HLSL non-arrayed sampler declarations	Restricted to ArraySize = 1	Generalized to any ArraySize (including > 2)

HLSL non-arrayed sampler declarations indicates allowing certain shaders that sample from single-subresource-resources currently to also sample from single-subresource-views of resources with multiple subresources.

Note that drivers must support extended array for Stereo 3D APIs, but for fullscreen exclusive Stereo 3D apps to display, the driver should also support Stereo scanout on the output. For example, in a multi-adapter system with two WDDM 1.2 graphics adapters, either adapter may be used to create a stereo windowed swapchain, even if only one of the adapters actually supports stereo output.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D111Core

D3D 11.1 core feature

In this topic:

Device.Graphics.AdapterRender.D3D111Core.D3D111CorePrimary

If a graphics device supports Direct3D 11.1, it must comply with the Direct3D 11.1 and DXGI Specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A graphics device must meet all requirements defined in the Direct3D 11.1 specification and must provide sufficient performance for Direct3D 11.1 features.

Direct3D 11.1 is a superset of Direct3D 11 (which is a strict superset of Direct3D 10.1 and Direct3D 10); therefore, implementation of Direct3D 11.1 also requires full support for the features defined by the Direct3D 10, Direct3D 10.1 and Direct3D 11 specifications respectively.

All features required by this specification must be exposed by the device driver, including those features defined for the DXGI DDI header.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11ASTC

In this topic:

Device.Graphics.AdapterRender.D3D11ASTC.CoreRequirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

D3D11 drivers that implement ASTC LDR support must support sampling from all 2D ASTC formats. ASTC formats are not individually supportable, and thus must all be support for ASTC to be considered supported. Here are a list of all ASTC formats that must be support for ASTC LDR Support to be valid: (All formats have _TYPELESS, and _SRGB equivalent)

- DXGI_FORMAT_4X4
- DXGI_FORMAT_5X4
- DXGI_FORMAT_5X5
- DXGI_FORMAT_6X5
- DXGI_FORMAT_6X6
- DXGI_FORMAT_8X5
- DXGI_FORMAT_8X6
- DXGI_FORMAT_8X8
- DXGI_FORMAT_10X5
- DXGI_FORMAT_10X6
- DXGI_FORMAT_10X8
- DXGI_FORMAT_10X10
- DXGI_FORMAT_12X10
- DXGI_FORMAT_12X12

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11ConservativeRasterization

In this topic:

Device.Graphics.AdapterRender.D3D11ConservativeRasterization.CoreRequirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D 11 should implement DDIs related to Conservative Rasterization, with allowance for tiers of functionality:

- Uncertainty Range:
- Inner Coverage
- Post-snap degenerate culling
- Other Pipeline Interactions:
  - TIR
  - MSAA
  - SampleMask w/ Inner Coverage
  - Clip Distance
  - Queries
  - oMask
  - HelperPixel coverage
  - Attribute Interpolation (extrapolation)
  - Early Z w/ Depth Extrapolation

For additional details, refer to the Conservative Rasterization spec.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11Core

D3D 11 core feature

In this topic:

Device.Graphics.AdapterRender.D3D11Core.D3D11CorePrimary

If a graphics device implements Direct3D 11, it must comply with the Direct3D 11 and DXGI Specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a graphics device implements Direct3D 11, it must meet all requirements defined in the Direct3D 11 specification and must provide sufficient performance for Direct3D 11 features.

Since Direct3D 11 is a superset of Direct3D 10, implementation of Direct3D 11 also requires support of Direct3D 10.1 feature set and by extension Direct3D 10.

All features required by this specification must be exposed by the device driver including those features defined for the DXGI DDI header.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11DoublePrecisionShader

D3D11* Double Precision Shader Functionality

In this topic:

Device.Graphics.AdapterRender.D3D11DoublePrecisionShader.D3D11CoreC

If the graphics hardware implements D3D11* Double Precision, it must conform to the feature specification as outlined in the D3D11 hardware specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D11 Specification allows for optionally implementing Double Precision Shader functionality on D3D11* hardware. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D11 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11DriverCommandLists

D3D11* Driver Command Lists

In this topic:

Device.Graphics.AdapterRender.D3D11DriverCommandLists.D3D11CoreB

If the graphics hardware implements the D3D11* driver command list, it must conform to the feature specification as defined in the D3D11 hardware specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D11 Specification allows for optionally implementing DriverCommandListfunctionality on D3D11* hardware. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D11 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11DriverConcurrentObjectCreation

D3D11* Driver Concurrent Object Creation

In this topic:

Device.Graphics.AdapterRender.D3D11DriverConcurrentObjectCreation.D3D11CoreA

If the graphics hardware implements the D3D11* Driver Concurrent Object Creation, it must conform to the feature specification as defined in the D3D11 hardware specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D11 Specification allows for optionally implementing Driver Concurrent Object creation functionality on D3D11* hardware. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D11 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11Level9WDDM12

WDDM 1.2 updates to the D3D9 UM DDI

In this topic:

Device.Graphics.AdapterRender.D3D11Level9WDDM12.D3D9UMDDIUpdate

If the graphics hardware driver implements the WDDM1.2 specification, it must include the D3D9 User Mode DDI additions as defined by the D3D11.1 API/DDI Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A WDDM 1.2 graphics driver is required to implement the D3D9 Adapter DDI and D3D9 DDI additions as defined in the D3D11.1 API/DDI specification in addition to the D3D9 DDI, as defined by the DirectX 9 hardware and driver specifications.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11Level9WDDM13

WDDM1.3 updates to the D3D9 UM DDI

In this topic:

Device.Graphics.AdapterRender.D3D11Level9WDDM13.LargeCaptureTextures

Direct3D Large Capture Textures

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All 9 UMDs, regardless of maximum feature level of hardware, must now properly validate the texture size parameters being passed to CreateResource2.

Graphics drivers must now gracefully fail (by returning a E_INVALIDARG) CAPTURE texture create requests that exceed the capabilities of the hardware.

Any CAPTURE textures that are approved for creation must behave identical to CAPTURE textures as they are defined.

Device.Graphics.AdapterRender.D3D11Level9WDDM13.Level9Instancing

Level 9 Instancing

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

In Windows, the OS will be making increased use of D3D Instancing to reduce CPU/GPU usage. All WDDM1.3 drivers must support Instancing as described in the Feature Level 9_3.

Device.Graphics.AdapterRender.D3D11Level9WDDM13.NativeStagingBuffers

Level9 Native Staging Buffer Performance Validation

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Directx9 level graphics hardware that implement WDDM1.3 must support the D3D9 Native Staging buffers DDI.

Device.Graphics.AdapterRender.D3D11Level9WDDM13.NativeUpdateSubresource

Direct3D Level9 Native UpdateSubresource

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

For Windows, the Direct3D9 DDI spec will include native support for the UpdateSubresource DDI. When D3D9-level parts implement this DDI, the UpdateSubresource API calls for a given amount of memory must be executed no slower than a CPU copy operation.

Device.Graphics.AdapterRender.D3D11Level9WDDM13.TimestampCounterSupport

Direct3D Level9 Timestamps and Counters

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Timestamp Query support will be made mandatory for all 9-level WDDM 1.3 drivers, specifically these Query types:

- D3DDDIQUERYTYPE_TIMESTAMP
- D3DDDIQUERYTYPE_TIMESTAMPDISJOINT
- D3DDDIQUERYTYPE_TIMESTAMPFREQ

Additionally, the CheckDounter and CheckCounterInfo Direct3D11 Counter DDIs must be supported.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11PartialPrecision

D3D11 Partial Precision shader support

In this topic:

Device.Graphics.AdapterRender.D3D11PartialPrecision.D3D11CoreE

If the graphic hardware implements the D3D11.1 Partial Precision Shader Functionality, it must conform to the feature specification as defined in the D3D11.1 hardware specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D11.1 Specification allows for optionally implementing Partial Precision Shader functionality on D3D9, D3D10*, and D3D11* hardware with a WDDM 1.2 driver. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D11.1 Hardware Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11RasterizerOrderedViews

In this topic:

Device.Graphics.AdapterRender.D3D11RasterizerOrderedViews.CoreRequirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D11 should implement DDIs related to Raster Order Views:

- Correct ordering of operations on all RasterizerOrdered* types

For additional details, refer to the ROV spec.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11StencilReference

In this topic:

Device.Graphics.AdapterRender.D3D11StencilReference.CoreRequirement[If implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D11 should implement DDIs related to Pixel Shader Stencil Reference:

- Writes to SV_STENCILREF function as expected

For additional information, refer to the PS-Specified Stencil Ref spec.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11TypedUAVLoad

In this topic:

Device.Graphics.AdapterRender.D3D11TypedUAVLoad.CoreRequirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D11 should implement DDIs related to Typed UAV Loads:

- Support typed load for all required UAV formats if typed UAV load flag is set
- Support typed load for optionally declared types

For more information, refer to the Typed UAV Loads spec

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11WDDM12

D3D 11 core feature with WDDM 1.2 additions

In this topic:

Device.Graphics.AdapterRender.D3D11WDDM12.D3D11v12Primary

If a WDDM 1.2 graphics device implements Direct3D 11, it must comply with the Direct3D 11, DXGI, and the D3D10 portion of the Direct3D 11.1 Features Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a graphics device implements Direct3D 11, it must meet all requirements defined in the Direct3D 11 specification and must provide sufficient performance for Direct3D 11 features.

Since Direct3D 11 is a superset of Direct3D 10, the implementation of Direct3D 11 also requires support of the Direct3D 10.1 feature set and by extension Direct3D 10.

In Windows, the following features originally defined in the D3D9 Hardware specification are now also required:

- Half Precision pixel formats (5551, 565, 4444)
- Same Surface Blts

Please see the Direct3D 11.1 Features Spec for complete details of all new features required to be exposed with a WDDM 1.2 driver for D3D10+ hardware.

All features required by this specification must be exposed by the device driver, including those features defined for the DXGI DDI header.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11WDDM12DoublePrecisionShader

D3D11* Double Precision Shader Functionality with additional ops codes introduced with WDDM 1.2

In this topic:

Device.Graphics.AdapterRender.D3D11WDDM12DoublePrecisionShader.D3D11v12C

If the graphics hardware implements the D3D11* Double Precision Shader Functionality with WDDM 1.2 driver additions, it must conform to the feature specifications as defined in the D3D11 hardware specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The D3D11 Specification allows for optionally implementing Double Precision Shader functionality on D3D11* hardware. If the hardware includes support for and the driver exposes this functionality, it must conform to the specifications for this feature as defined in the D3D11 Hardware Specification. For Windows, if a WDDM 1.2 Graphics device driver supports Double Precision Shader functionality, it is required to support the extended double precision math as described in the Shader Model Improvements Specification.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11WDDM13

D3D11.1 Tiled Resource support

In this topic:

Device.Graphics.AdapterRender.D3D11WDDM13.MapDefault

Map Default

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

For Direct3D Feature Level 11_0 and higher parts, the WDDM1.3 drivers must allow mappable DEFAULT buffers to be created. Apps will be able to request these mappable resources by setting the CPU_Read/CPU_Write access flags as described in the WDDM1.3 spec. The mappable DEFAULT buffers should behave identically to existing DEFAULT buffers today from an API perspective (other than being able to be mapped). The mapping/CPU-access performance of mappable DEFAULT buffers should be comparable to STAGING buffers.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D11WDDM20

In this topic:

Device.Graphics.AdapterRender.D3D11WDDM20.CoreRequirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

- High Performance Timing Data:

Note: This feature is Mandatory for a WDDM 2.0 driver supporting D3D11.

“High Performance Graphics Timing Data” provides light weight and highly detailed timing data for graphics workloads. This data is used by analysis tools to identify performance or other graphics related issues in Windows applications or the graphics stack.

A WDDM2.0 driver must ensure that the graphics device/driver provides the following:

A high resolution GPU Timer

12.5 MHz (80ns resolution) or better.

At least 32 bits of timestamp resolution

The GPU timestamp can be sampled for all engines in a GPU

The GPU timestamp can be sampled at the end of the GPU pipeline

The GPU timestamp frequency can be sampled.

The GPU timestamp is invariant and is unaffected by p-state transitions.

Among the other changes for this feature, this will include the addition of two new flags to the existing DdiControlEtwLogging interface; when these flags are set so that the first flag is value of 1 and the second flag is a value of 0, then the driver must ensure that:

All engine components must ensure that they are never clock- or power-gated as long as the flag remains enabled, and must in general refrain from entering any idle states. The components must remain active to ensure there is no latency added due to power transitions.

All engine components must ensure that their processing frequencies and functional bus bandwidths are kept at their maximum stable operating values. Thermal events requiring P-State transition down should still occur to prevent damage to the hardware, but P-States should be defined so that these are exceptional occurrences that are not normally seen in cool lab environments.

The sampled GPU timestamp can be correlated with previously issued graphics commands

Generation of timing data is off by default, but can be turned on and off at any time.

The overhead of collecting this performance data is no slower than using the timestamp query technique that is already available in D3D9 and D3D10+

Timing data can be collected in direct command lists and bundles. Timing data can be enabled/disabled on a per-command list basis.

Resulting data on Tile based deferred rendering hardware appears as if it was generated on an immediate mode rendering device.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D12ASTC

In this topic:

Device.Graphics.AdapterRender.D3D12ASTC.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D12 must meet all requirements with sufficient performance defined in the Direct3D 12 Specifications.

D3D12 drivers that implement ASTC LDR support must support sampling from all 2D ASTC formats. ASTC formats are not individually supportable, and thus must all be support for ASTC to be considered supported. Here are a list of all ASTC formats that must be support for ASTC LDR Support to be valid: (All formats have _TYPELESS, and _SRGB equilovants.)

- DXGI_FORMAT_4X4
- DXGI_FORMAT_5X4
- DXGI_FORMAT_5X5
- DXGI_FORMAT_6X5
- DXGI_FORMAT_6X6
- DXGI_FORMAT_8X5
- DXGI_FORMAT_8X6
- DXGI_FORMAT_8X8
- DXGI_FORMAT_10X5
- DXGI_FORMAT_10X6

- DXGI_FORMAT_10X8
- DXGI_FORMAT_10X10
- DXGI_FORMAT_12X10
- DXGI_FORMAT_12X12

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D12ConservativeRasterization

In this topic:

Device.Graphics.AdapterRender.D3D12ConservativeRasterization.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D12 must implement DDIs related to Conservative Rasterization, with allowance for tiers of functionality:

- Uncertainty Range:
- Inner Coverage
- Post-snap degenerate culling
- Other Pipeline Interactions:
- TIR

- MSAA
- SampleMask w/ Inner Coverage
- Clip Distance
- Queries
- oMask
- HelperPixel coverage
- Attribute Interpolation (extrapolation)
- Early Z w/ Depth Extrapolation

For additional details, refer to the Conservative Rasterization spec.

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D12Core

D3D 12 core feature

In this topic:

Device.Graphics.AdapterRender.D3D12Core.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D12 must meet all requirements defined in the **Direct3D 12 Specifications**.

- **User Heaps**

The GPU must read and write data to and from memory pages. When such pages are accessible by the CPU, they must be able to be marked as write-back or write-combine for the CPU.

The driver must honor write-combine requests to efficiently operate with other adapters and engines on the system. GPUs must fully support the other page properties orthogonally,.

The GPU must be able to read and write single-dimensional buffer data & multi-dimensional texture data to and from such pages for all GPU operations. However, restrictions and undefined behavior exists and is explained in the specification.

It is acceptable for resources and heaps to be mapped while the GPU is actively reading or writing from such pages. Please refer the DDI specification for details on the GPU read write operations

GPUs capable of I/O coherence must use it when write-back is required or chosen for pages that reside in physical system memory.

Data typically associated with buffers and pitch-linear texture data operations are required to be aligned properly for consumption by the GPU. Both buffer and pitch-linear data may reside next to each other and even overlap.

- Pitch-linear requirements & restrictions are as follows:
  - Only a single 3D subresource, at a time, is required of pitch-linear/ row-major copying.
  - Only a single 2D subresource, at a time, is required of pitch-linear/ row-major sampling, rendering.
  - The base address/ offset of pitch-linear texture data must be 512-byte aligned or less.
  - The width stride must be 128-byte aligned, or less, for all texel element sizes. The width stride is 32-bit and can be much greater than the tightest stride that is correctly aligned for a particular texture width.
  - Arbitrary height, up to the D3D maximum, is supported orthogonally.
  - For volume textures, the depth stride is height times width stride.
  - Compressed render targets must be able to be resolved to enable the CPU to read/ write the texels.
- Buffer alignment restrictions are not changed over D3D11:

- Constant data reads must be a multiple of 256 bytes from the beginning of the heap (i.e. only from addresses that are 256-byte aligned), or less.
- Index data reads must be a multiple of the index data type size (i.e. only from addresses that are naturally aligned for the data), or less.
- Draw*Indirect data must be from offsets that are multiples of 4 (i.e. only from addresses that are DWORD aligned), or less.

- When the GPU supports standard swizzle, the GPU must support all multi-dimensional operations as orthogonally as other textures, such as texture from, render to, copy to & from, re-swizzle to & from, etc.
  - Element size for block-compressed formats and ASTC formats must be same as the block size.
  - GPU must support standard swizzle from all engines, including scan-out.
  - Compressed render targets must be able to be resolved to enable the CPU to read/ write the texels.
  - Standard swizzle data must be aligned to 4KB and 64KB page boundaries.
- The GPU must not require texture alignment greater than 64KB, except for MSAA resources with sample count greater than 1.
- Support page-based virtual addressing of physical memory from all engines on the GPU.
  - When the GPU supports GPU virtual addressing with proper security boundaries to allow user mode modification, it must be supported from all engines to realize its full benefit.
- **Resource Binding:**

  Display drivers for the Direct3D 12 API must implement DDIs related to object descriptors and descriptor heap (as follows), with allowance for tiers of functionality:
  - Capability Query about Resource Binding Tiers
  - Descriptor Heaps
  - Setting Descriptor Heaps
  - Creating Descriptors
  - Copying Descriptors

- Creating a Root Table Layout
- Setting a Root Table Layout
- Setting Descriptor Tables on the Root Table
- Setting Constants on the Root Table
- Setting Descriptors on the Root Table (Bypassing Descriptor Heap/Tables)
- Setting IA/VB/SO/RT/DS Descriptors On A Command List / Bundle
- View Manipulation

Please see sections of “Levels of Hardware Support” and “DDI” in the D3D12 Resource Binding - Functional Spec for more details.

- **High Performance Timing Data:**

  “High Performance Graphics Timing Data” provides light weight and highly detailed timing data for graphics workloads. This data is used by analysis tools to identify performance or other graphics related issues in Windows applications or the graphics stack.

  - A WDDM1.3 driver must ensure that the graphics device/driver provides the following:
  - A high resolution GPU Timer
  - 12.5 MHz (80ns resolution) or better.
  - At least 32 bits of timestamp resolution
  - The GPU timestamp can be sampled for all engines in a GPU
  - The GPU timestamp can be sampled at the end of the GPU pipeline
  - The GPU timestamp frequency can be sampled.
  - The GPU timestamp is invariant and is unaffected by p-state transitions.
  - Among the other changes for this feature, this will include the addition of two new flags to the existing DdiControlEtwLogging interface; when these flags are set so that the first flag is value of 1 and the second flag is a value of 0, then the driver must ensure that:
  - All engine components must ensure that they are never clock- or power-gated as long as the flag remains enabled, and must in general refrain from entering any idle states. The

components must remain active to ensure there is no latency added due to power transitions.

- All engine components must ensure that their processing frequencies and functional bus bandwidths are kept at their maximum stable operating values. Thermal events requiring P-State transition down should still occur to prevent damage to the hardware, but P-States should be defined so that these are exceptional occurrences that are not normally seen in cool lab environments.
- The sampled GPU timestamp can be correlated with previously issued graphics commands
- Generation of timing data is off by default, but can be turned on and off at any time.
- The overhead of collecting this performance data is no slower than using the timestamp query technique that is already available in D3D9 and D3D10+
- Timing data can be collected in direct command lists and bundles. Timing data can be enabled/disabled on a per-command list basis.
- Resulting data on Tile based deferred rendering hardware appears as if it was generated on an immediate mode rendering device.

- **Timestamp Queries:**
  - A straightforward mapping of D3D11 UAV counters, stream-output counters, and queries to the D3D12 API constructs must produce results that pass the relevant D3D11 HLK tests.
  - D3D12 drivers must implement the new binary occlusion query correctly.
  - A GPU page fault must occur when a shader accesses a non-existent UAV counter.
  - Shader dynamic indexing of UAV counters must work correctly
  - Timestamp queries must work on 3D and compute command queues.
  - Predication must work correctly for all rendering operations (including those contained in a bundle)
  - Stream-output, UAV counters, and predication buffers must work correctly with all D3D12 resource heap types.
- **Indirect Rendering:**
  - A D3D12 driver must implement GPU drawing/dispatch from indirect agument buffers.

- Rendering with an aribtrary indirect argument buffer should produce the same results as the equivalent set of D3D12 command list API calls.
- Indirect argument buffers must be supported in all D3D12 heap types.
- Aribtrary byte strides (which are multiples of 4) between per-draw structures must be supported.
- A single command list which both generates an indirect argument buffer and consumes it must work correctly (drivers must synchronize properly).
- D3D12 drivers must program the GPU to compute the minimum of the MaxDrawCount passed to the driver, and the DrawCount passed in an indirect argument buffer.

- **Pipeline State Caching:**
  - A D3D12 driver must implement DDIs for retrieving hardware-native shader code for a pipeline state object, and DDIs for reconstructing pipeline states from this cached shader code.
  - Any pipeline state that can be constructed at the API must be cacheable, including compute pipelines.
  - Using a pipeline state constructed from cached shader code must produce identical rendering results compared to one produced from HLSL-compiled bytecode.

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D12Multiadapter

In this topic:

Device.Graphics.AdapterRender.D3D12Multiadapter.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D12 must meet all requirements with sufficient performance defined in the **Direct3D 12 Specifications**.

D3D12 display drivers must support cross-adapter shared surfaces as described in the Direct3D 12 Specifications.

If the driver runs on linked display adapters then the Direct3D 12 linked adapter functionality is required.

Note: This feature is Mandatory for a driver claiming to be D3D12 compliant driver.

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D12RasterizerOrderedViews

In this topic:

Device.Graphics.AdapterRender.D3D12RasterizerOrderedViews.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D12 must implement DDIs related to Raster Order Views:

- Correct ordering of operations on all RasterizerOrdered* types

For additional details, refer to the ROV spec.

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D12StencilReference

In this topic:

Device.Graphics.AdapterRender.D3D12StencilReference.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D12 must implement DDIs related to Pixel Shader Stencil Reference:

- Writes to SV_STENCILREF function as expected

For additional information, refer to the PS-Specified Stencil Ref spec.

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D3D12TypedUAVLoad

In this topic:

Device.Graphics.AdapterRender.D3D12TypedUAVLoad.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Display drivers for Direct3D12 must implement DDIs related to Typed UAV Loads:

- Support typed load for all required UAV formats if typed UAV load flag is set
- Support typed load for optionally declared types

For more information, refer to the Typed UAV Loads spec.

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.AdapterRender.D312VolumeTiledResources

In this topic:

Device.Graphics.AdapterRender.D3D12VolumeTiledResources.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

Drivers reporting support for Volume Tiled Resources (via exposing Tiled Resources Tier 3) must conform to the specifications for this feature outlined in the D3D IHV specs. An example of an interesting spec detail for Volume Tiled Resources is the required tile shapes for various surface formats when used in a Volume Tiled Resource. All other behaviors for this feature fall under the general specifications for tiled resources in general, as well as volume textures in general – as this feature is the intersection of the two.

Business Justification:

All display drivers which implement the D3D12 DDI must do so in a consistent manner. This enables the D3D12 API to have a consistent behavior across many platforms, which reduces the cost to ISVs of developing 3D applications on PC/Tablet/Phone/XBOX platforms.

Send comments about this topic to Microsoft

Device.Graphics.WDDM

The base feature set implemented by drivers supporting all versions of the WDDM.

In this topic:

Device.Graphics.WDDM.Base

Graphics drivers must be implemented per the WDDM 1.0 spec.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

As implied by the WDDM 1.0 Specification, display drivers must minimally support D3D9 and Pixel Shader 2.0.

WDDM 1.0 introduces the following key requirements:

- User-mode Display Driver

- Video Memory Manager
  - (p1) Linear Memory Manager
  - (p1) Rectangular Memory Manager
- GPU Scheduler
- Mode-Switch Architecture Cleanup
- Merged Miniport and DLL
- Recovery from Hardware Hangs
- Simplified Kernel-mode Objects
- Legacy DDI Consolidation
- Hot-plug of Display Cards, Hot Docking, and Support for "Clone View"

MSDN documentation is updated based on the WDDM 1.0 Specification. Please verify MSDN documentation for WDDM 1.0 requirements.

Device.Graphics.WDDM.Checklist

All graphics devices must comply with base requirements checklist for graphics cards, chipsets, and drivers.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All graphics cards must adhere to the following checklist:

Memory Allocations and Access (applicable for all form factors)

- The GPU must not access an allocation after the last DMA buffer that was submitted to the GPU referencing that allocation, is reported as complete.
- The GPU must not access any allocations that were not specified in the allocation list associated with the DMA buffer being executed.

Card/Chipset Requirements (not applicable for server devices)

- For a multi-headed display adapter, it is recommended that the display adapter expose all display resources through a single function and not as a multifunction device. If a sound controller or tuner is part of the device, the device can then be exposed as a multifunction device.
    - If a second display class function is exposed for legacy compatibility, the adapter must be fully functional without using the second head. The second (or additional) functional heads must:
        - Have sub-class 80 (other) to avoid a generic driver being used.
        - Not have an expansion ROM.
        - Not describe more than 1 MB of total memory space resources.
        - Not duplicate the frame buffer resources.
        - Not describe any interrupt resources.
        - Not describe any I/O space resources.
        - Non display base class functions on the same device, such as a multimedia device class, sub class video device, or sub class audio device, are not subject to these restrictions.

WDDM Driver Checklist (not applicable for non-WDDM drivers)

- WDDM drivers must not report a submission fence as completed until the operation for the associated submission is truly completed. This is required by the scheduling model in WDDM. The WDDM must not use the DDI in such a way that the stability of Windows is compromised.
- WDDM drivers must insert a fence/interrupt pair for each fence requested and must not hold off reporting the completion of that fence. The fence must be reported as soon as the associated required interrupt is generated by the GPU. For example, it must not wait until the timer interrupt or until the next VSync. This is required by the Scheduling model in DDM.
- WDDM drivers must not wait or block during DdiSubmitCommand, which is necessary from the perspective of the Video Scheduler in WDDM. The driver must not wait or block during a DdiBuildPagingBuffer call as well.
- The WDDM driver must not expose more than one node per physical engine. The driver and GPU cannot schedule a single physical engine between multiple nodes.

- The WDDM driver must not map a virtual address to video memory that is directly exposed to an application. This is fundamental to the implementation of video memory management support in the WDDM driver. The WDDM driver must not hide or expose video memory in a way that the video memory manager is unaware of. Exceptions to this are allowed specifically for the implementation of GPU Developer Tools (Debuggers, Profilers, …). Such an exception must apply only in a scenario where the GPU developer tools, in order to perform, need to map video memory to virtual address space, for the duration of the session and only for the process of the application being operated on.
- The WDDM driver must not expose any memory segments that are used for the sole purpose of reporting additional video memory than is actually present for its appropriate use. The correct amount of video memory must be reported for use by various applications through Windows. The WDDM driver can use up to 5% of the system memory for internal use such as cursor bitmaps, ring buffer, etc; any amount above this must not be used to hide or expose video memory in a fashion such that the video memory manager is unaware of.
- A WDDM driver must use ACPI for all interactions with the system BIOS. SMI is currently allowed, but is highly discouraged by Microsoft. See WinHEC 2005 presentation, TWAR05007.

Design Notes:

For more information on any of the items in the Details section, refer to the Windows Driver Kit and search for the relevant keywords.

Device.Graphics.WDDM.GPUFenceCommands

GPU that is capable of processing fence commands in the command queue must trigger an interrupt to the CPU when it consumes a fence command.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

When the hardware consumes a fence command, it must notify the operating system by triggering an interrupt to the CPU, with the fence ID communicated to the ISR.

Design Notes:

See the Windows Driver Kit

Send comments about this topic to Microsoft

Device.Graphics.WDDM.Display

The base feature set implemented by drivers supporting all versions of the WDDM Display DDIs.

In this topic:

Device.Graphics.WDDM.Display.Base

Graphics drivers must be implemented per the WDDM 1.0 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

See requirement **Device.Graphics.WDDM.Base**

Device.Graphics.WDDM.Display.GammaCorrection

A graphics adapter must support gamma correction in hardware without using any additional graphics memory bandwidth.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

ICM uses the ability to perform gamma correction for the attached monitor and to allow game applications to switch palettes. This capability also supports transition effects in applications. To support ICM, the display adapter or chipset gamma must be programmatically adjustable.

To perform gamma correction in hardware, downloadable RAM DAC entries (LUT) must be included.

The LUT must implement at least 256 entries per component input for 8-bit color channel components. Hardware may implement the LUT for larger component resolutions by using interpolation if at least 128 sample points are used.

This ability must be supported without requiring the use of graphics memory bandwidth.

Device.Graphics.WDDM.Display.HotPlugDetection

A graphics adapter must reliably detect the connect and disconnect event of display devices.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows supports two levels of display detection - interruptible and poll-able.

- Interruptible is defined as the case where the graphics adapter is able to automatically detect a change in display connectivity and report it to Windows. The ability of the hardware to automatically generate an interrupt for display connectivity change is called Hot Plug Detect (HPD). The HPD must not cause any visual corruption on any display already connected to the system.
- Poll-able is defined as the case where Windows has to explicitly query the graphics adapter to query for a change in the display connectivity. In some cases, visual corruption might be displayed for a very brief time.

All standard digital connectors (DisplayPort, HDMI, DVI) support HPD and the graphics adapter must report these connectors as interruptible. Once the hardware generates the interrupt, the graphics adapter must notify Windows via the DxgkCbIndicateChildStatus DDI found here: http://msdn.microsoft.com/en-us/library/ff559522(VS.85).aspx.

Analog connectors (HD-15, S-Video, Component, Composite) are not required to support interruptible display detection. However, it is possible that HPD can be implemented in the hardware (e.g. load sensing, I2C) for such connectors. In such a case, the graphics adapter must report this connector as interruptible as above. Software polling cannot be used to achieve HPD functionality for analog connectors.

For those analog connectors, where HPD is not implemented in the hardware, the graphics adapter must report the connector as polled. In such a case, the graphics adapter must only perform detection on the connector when explicitly requested by Windows via the D3DKMTPollDisplayChildren DDI, found here: http://msdn.microsoft.com/en-us/library/ff547077(v=VS.85).aspx.

Design Notes:

See the Windows Driver Kit: http://msdn.microsoft.com/en-us/library/ff559522(VS.85).aspx for " DxgkCbIndicateChildStatus."

The following HPD methods are VESA standards: DVI HPD is covered in the VESA Plug and Play (PnP) Standard for the Display/Graphics Subsystem, Release A. DisplayPort HPD is covered in all versions of the DisplayPort standard.

Device.Graphics.WDDM.Display.I2CSupport

A graphics device driver must have I2C support in WDDM.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following is the set of interfaces that every WDDM driver is required to implement for I2C support:

DxgkDdiI2CReceiveDataFromDisplay

DxgkDdiI2CTransmitDataToDisplay

These interfaces are documented in the WDK and can be found here: http://msdn.microsoft.com/en-us/library/ff567386(v=VS.85).aspx.

Device.Graphics.WDDM.Display.Multimon

If a graphics adapter supports more than 1 source and 1 target, it must support all multiple-monitor configurations in Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A graphics driver can enumerate sources and targets based on its capabilities. Windows queries the driver for the number of sources and targets by calling the DxgkDdiEnumVidPnCofuncModality DDI, found here http://msdn.microsoft.com/en-us/library/ff559649.aspx. Windows supports the following monitor configurations:

- Single monitor configuration - Only one physical monitor is active and the entire desktop is displayed on it.

- Extended monitor configuration - Multiple monitors are active in and different parts of the desktop are displayed on it. GDI must be made aware of the monitor boundaries such that Windows features like maximize, aero snap etc work according to spec.
- Duplicate monitor configuration - The exact same desktop contents are displayed on multiple monitors.

The number of targets must always be greater than or equal to the number of sources.

If the driver enumerates exactly 1 source and 1 target, then no multiple monitor configurations are supported.

If the driver enumerates exactly 1 source and multiple targets, then the driver must support single monitor and duplicate monitor configurations.

If the driver enumerates multiple sources and multiple targets, then the driver must support all the supported configurations. Additionally:

- The capability of each source when enabled by itsself, with respect to resolution, Direct3D, protected content playback, should be the same. The capabilities of the target will vary based on the target type.
- The operating system must be able to drive any target from any source, although the driver can constrain which targets can be driven in combination.

Multiple-monitor support is built into Windows; therefore, graphics drivers must not include any special code to provide support already available in the OS.

It must be possible for a user to set all the configurations supported using the Windows Display Control Panel and by pressing the Win+P key combination.

Send comments about this topic to Microsoft

Device.Graphics.WDDM.Display.HDMIorDPDCNs

The optional feature implemented by WDDM drivers supporting the audio DCNs over HDMI or DisplayPort.

In this topic:

Device.Graphics.WDDM.Display.HDMIorDPDCNs.DCNCompliance

A display driver that contains either an HD Audio interface that supports multi-channel HDMI or a DisplayPort audio consistent with HD Audio must comply with HD Audio HDMI & DisplayPort DCNs.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a display driver is designed for an HDMI or DisplayPort adapter or chipset that contains an HD Audio interface that implements any of the **verbs listed below**, the graphics driver must:

- Comply with the following HD Audio DCNs:
  - HDA034-A2: HDMI Content Protection and Multi-Channel Support
  - HDA039-A: HDMI/ELD Memory Structure
  - HDA036-A: DisplayPort Support and HDMI Miscellaneous Corrections
- Read and parse the EDID from any attached HDMI or DisplayPort sink device and populate the ELD buffer to accurately reflect the hardware and sink device capabilities as required in the HD Audio DCN HDA039-A: HDMI/ELD Memory Structure.
- Program all possible Short Audio Descriptors (SADs) from the EDID into the ELD.
- Correctly program the Presence Detect (PD) and ELD Valid (ELDV) bits on the HDMI or DP transmitter hardware for consumption by the audio driver in response to the following events as outlined in the HD Audio DCN HDA034-A2: HDMI Content Protection and Multi-Channel Support.
  - Hot plug events (HDMI/DisplayPort Sink Connected/Disconnected)
    - According to DCN referenced above
  - Video mode changes
    - According to DCN referenced above
  - Graphics power state transitions
    - When the graphics subsystem exits power state D0:
    - If sink is attached, PD = 1, ELDV = 0
    - Otherwise, PD = 0, ELDV = 0
    - When graphics subsystem enters power state D0:

        - If sink is attached, PD = 1, ELDV = 1
        - Otherwise, PD = 0, ELDV = 0
    - Driver load
        - According to DCN referenced above
    - Driver unload
        - PD = 1, ELDV = 1
        - ELD_Version = 1Fh; indicative of basic audio
- Respond to HD Audio-initiated requests for HDCP as outlined in the HD Audio DCN HDA034-A2: HDMI Content Protection and Multi-Channel Support within 10 seconds.
- Ensure that the READY and CES (current encryption state) values in the CP_CONTROL verb accurately reflect the state of the display subsystem, as outlined in the HD Audio DCN HDA034-A2: HDMI Content Protection and Multi-Channel Support.

The Verbs are:

- F2Fh (Get HDMI ELD Data)
- F2Dh (Get Converter Channel Count)
- 72Dh (Set Converter Channel Count)
- F2Eh (Get HDMI Data Island Packet - Size Info)
- 72Eh (Set HDMI Data Island Packet - Size Info)
- F30h (Get HDMI Data Island Packet - Index)
- 730h (Set HDMI Data Island Packet - Index)
- F31h (Get HDMI Data Island Packet - Data)
- 731h (Set HDMI Data Island Packet - Data)
- F32h (Get HDMI Data Island Packet - Transmit-Control)
- 732h (Set HDMI Data Island Packet - Transmit-3Control)
- F33h (Get Content Protection Control)
- 733h (Set Content Protection Control)

- F34h (Get Converter Channel to HDMI Slot Mapping)
- 734h (Set Converter Channel to HDMI Slot Mapping)

Send comments about this topic to Microsoft

Device.Graphics.WDDM.DisplayRender

The base feature set implemented by drivers supporting all versions of the WDDM for both Display and Render DDIs.

In this topic:

Device.Graphics.WDDM.DisplayRender.Base

Graphics drivers must be implemented per the WDDM 1.0 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

See requirement **Device.Graphics.WDDM.Base**

Device.Graphics.WDDM.DisplayRender.DriverSetupCompatible

Graphics drivers must be implemented per the WDDM 1.0 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All WDDM graphics drivers being submitted for posting to Windows Update targeting Windows Client SKUs must install correctly on injection into the Windows OS image driver store during OS setup.

Device.Graphics.WDDM.DisplayRender.OutputProtection

A display adapter must support output connectors with content protection features and provides control via Protected Media Path-Output Protection Manager (PVP-OPM) and Certified Output Protection Protocol (COPP).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To enable the playback of premium video content, digital video outputs must support a digital monitor link protection mechanism such as HDCP to obtain Windows 8 Client Logo. This requirement is applicable for all full graphics devices.

The OPM API and Media Foundation PMP require display drivers to properly implement the OPM DDIs to handle premium content. High grade premium content will not be passed to the display driver unless the display driver has a PVP-OPM certificate. Drivers that support the OPM DDIs must have a driver certificate to testify that the compliance rules, robustness rules, and terms of the PVP-OPM legal agreement, have been met.

The display driver must support both the COPP* and PVP-OPM driver interfaces for controlling and signaling video output protection state.

Output protection behaviors are specified by the PVP-OPM compliance rules. The document is available to graphics vendors by request to wmla@microsoft.com.

The WDDM driver must implement the necessary Display Mode Management DDIs and structures as documented in the Windows Driver Kit and the WDDM documentation to enable TV playback and Analog Content Protection (ACP) support for Media Center*.

D3DKMDT_VIDPN_PRESENT_PATH_CONTENT: Media center will use this information to change the TV mode (TV_PLAYBACK vs. WIN_GRAPHICS)

D3DKMDT_VIDPN_PRESENT_PATH_COPYPROTECTION_TYPE and D3DKMDT_VIDPN_PRESENT_PATH_COPYPROTECTION_SUPPORT: These structures are necessary to provide ACP support through the LDDM driver.

S-Video and composite video output interfaces must support the following:

CGMS-A on Line 20 as specified by IEC 61880

CGMS-A on Line 21 as specified by EIA-608-B

Component (YPbPr) outputs must support the following:

CGMS-A with redistribution control as specified by EIA-805

When the TV-out interface is enabled, the display driver must expose a 720x480 60-Hz display mode and/or a 720x576 50-Hz display mode. These display modes must be exposed by the video miniport and added to the default timings list for Windows applications to set when connected to a standard definition TV.

Supports SDTV Modes:

The WDDM miniport driver must set this parameter to TRUE for the video output to expose SDTV modes like NTSC, PAL, or SECAM.

Cable Ready systems with CableCARD support with digital video outputs (for example HDMI, or DP) must support a CableLabs approved digital monitor link protection mechanism such as HDCP.

Design Notes:

- S-Video and composite video output interfaces may be implemented through the same connector. If a proprietary interface is used, an adapter must be made available either included with the system or available for purchase at point of sale. A system or device that uses 7-pin S-Video connectors is not required to provide an adapter so long as the first four pins on the 7-pin connector are electrically compatible with a standard 4-pin S-Video connector. Microsoft recommends including SCART output when appropriate for the region of sale.
- * COPP, ACP, CGMS-A, analog TV-out, and SDTV support are required on x86 and x64 architectures and operating systems only.

Device.Graphics.WDDM.DisplayRender.Stability

All WDDM graphics drivers must not generate any hangs or faults under prolonged stress conditions.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Graphics drivers must function properly, and not generate any hangs or faults throughout the duration of stress testing as specified in the "4-hour WDDM Profile".

To "stress" a graphics driver, Comparative Reliability Analyzer for Software and Hardware (CRASH) launches and terminates other test applications to simulate real-world scenarios.

Send comments about this topic to Microsoft

Device.Graphics.WDDM.Render

The base feature set implemented by drivers supporting all versions of the WDDM Render DDIs.

In this topic:

Device.Graphics.WDDM.Render.Base

Graphics drivers must be implemented per the WDDM 1.0 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

See requirement **Device.Graphics.WDDM.Base**

Device.Graphics.WDDM.Render.VideoDecoding

Display drivers must support the DirectX VA 2.0 Video Decoder DDI.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Display WDDM drivers must support the following DXVA 2.0 Video Decoder DDI.

WDDM drivers must support the following DXVA modes:

- DXVA_ModeH264_VLD
- DXVA2_ModeVC1_D OR DXVADDI_ModeVC1_D2010

If hardware supports acceleration for MPEG2, WDDM drivers support at least one of the following sets of MPEG2 GUIDs:

- DXVA2_ModeMPEG2_VLD and DXVA2_ModeMPEG2_iDCT
- DXVA2_ModeMPEG2_VLD and DXVA2_ModeMPEG2_MoComp

- DXVA2_ModeMPEG2_iDCT
- DXVA2_ModeMPEG2and1_VLD

If hardware supports acceleration for H.265, it must support HEVC/H.265 modes:

- D3D11_DECODER_PROFILE_HEVC_VLD_MAIN
- D3D11_DECODER_PROFILE_HEVC_VLD_MAIN10 (optional if hardware does not support Main10 profile)

It is highly recommended for drivers to support D3D11_DECODER_PROFILE_HEVC_VLD_MAIN10, as Main 10 is very common format for premium HEVC content.

WDDM drivers must support DXGI_FORMAT_420_OPAQUE and DXGI_FORMAT_NV12 as decoder output formats. NV12 is recommended for scenarios requiring shader interop. For HEVC Main 10, drivers must support DXGI_FORMAT_P010 as decoder output format.

If the display adapter supports hardware-accelerated decode of H.264, it must support either the DXVA_ModeH264_MoComp GUID or the DXVA_ModeH264_VLD GUID.

If the display adapter support hardware accelerated decode of HEVC, it must support the DXVA_ModeHEVC_VLD_Main GUID.

Finally, WDDM drivers must support Standardized AES 128 H.264*** and MPEG2, if MPEGE2 acceleration is supported.

Design Notes:

*** Standardized AES 128 support is required on x86 and x64 architectures and operating systems only.

Device.Graphics.WDDM.Render.VideoProcessing [IF Implemented]

A display WDDM driver must support the DirectX VA 2.0 Video Processor DDI.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If implemented, display WDDM drivers must support the DXVA 2.0 Video Processor DDI and implement support for the following Video Processor Device GUIDs:

- DXVADDI_VideoProcProgressiveDevice
- DXVADDI_VideoProcBobDevice

The following DXVA2 video processor caps must be supported for each of the video processor device GUIDs:

- DXVADDI_VIDEOPROCESS_YUV2RGB
- DXVADDI_VIDEOPROCESS_YUV2RGBEXTENDED
- DXVADDI_VIDEOPROCESS_STRETCHX
- DXVADDI_VIDEOPROCESS_STRETCHY
- DXVADDI_VIDEOPROCESS_SUBRECTS
- DXVADDI_VIDEOPROCESS_SUBSTREAMS
- DXVA2_VideoProcess_SubStreamsExtended
- DXVA2_VideoProcess_Constriction

In addition, to support DXVADDI_VIDEOPROCESS_YUV2RGBEXTENDED caps, the following color parameters must be supported when color-space converting from a YUV surface to a RGB surface:

- D3DDDIARG_VIDEOPROCESSBLT.DestFormat.NominalRange
  - DXVADDI_NominalRange_Unknown (should be interpreted as 0_255 if a sophisticated algorithm is not implemented)
  - DXVADDI_NominalRange_0_255
  - DXVADDI_NominalRange_16_235
- D3DDDIARG_VIDEOPROCESSBLT.pSrcSurfaces[].SampleFormat.VideoTransferMatrix
  - DXVADDI_VideoTransferMatrix_Unknown (should be interpreted as BT601 unless the source surface is greater than 576 height in which case should be interpreted as BT709)
  - DXVADDI_VideoTransferMatrix_BT709
  - DXVADDI_VideoTransferMatrix_BT601

The following YUV formats must be supported as the video stream:

- YUY2 - 8-bit packed 4:2:2
- NV12 - 8-bit planar 4:2:0

If the video processor supports a 10-bit YUV subsampling, the following corresponding format must be supported:

- Y210 - 10-bit packed 4:2:2
- Y410 - 10-bit packed 4:4:4

- P210 - 10-bit planar 4:2:2
- P010 - 10-bit planar 4:2:0

If the video processor supports 16-bit YUV formats, the following formats must be supported:

- Y216 - 16-bit packed 4:2:2
- Y416 - 16-bit packed 4:4:4
- P216 - 16-bit planar 4:2:2
- P016 - 16-bit planar 4:2:0

The following YUV format must be supported as the video sub-streams:

- AYUV - 8-bit packed 4:4:4

For these formats, color converting YUV-to-RGB blits run through the VideoProcessBlt function must at least be able to use BT. 601 and BT. 709 conversion matrices. This process allows the graphics to switch between the YUV-to-RGB matrix transforms for different color formats, to ensure proper handling of video that originates from different standard color spaces such as those defined in ITU-R Recommendations BT. 601 and BT.709, is required.

Tolerance threshold: 50dB of quality difference between reference and DXVAHD modes

- Color conversion support of the following for playback, transcode and capture scenarios:
  - NV12->ARGB32
  - YUY2->ARGB32
  - ARGB32->NV12 (both full-swing and studio-swing)
  - 420O->NV12
  - 420O->ARGB32
  - AYUV->NV12
- Rescaling support for the above conversions
- Rotation support
- Extended Range (Full-Swing) support for transcode and capture scenarios

Support for updated DX9 and DX10+ user mode DDIs as documented on Connect at: https://connect.microsoft.com/site1304/Downloads/DownloadDetails.aspx?DownloadID=47236.

Send comments about this topic to Microsoft

Device.Graphics.WDDM11

The base feature set implemented by drivers supporting WDDM 1.1.

In this topic:

Device.Graphics.WDDM11.Base

A graphic driver that is written for a discrete graphic adapter or an integrated graphics adapter device must meet all requirements defined in the WDDM 1.1 specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM 1.1 introduces the following key requirements over WDDM 1.0:

- Hardware acceleration of select GDI features.
- Implementation of DMM DDIs for Connecting and Configuring Displays
- Support for 32-bit BGRA pixel format compatible with GDI. (Through DirectX10 and later DDIs.)
- Provide additional information to aid in debugging VSync TDRs.
- Kernel mode drivers must be compiled with Frame Pointer Optimizations (FPO) disabled.
- Optional features, if implemented, must be incorporated according to the specifications and WDK documentation, including Standardized AES128, DXVA-HD, overlays, and Direct3D11.

MSDN documentation is updated based on the WDDM 1.1 Specification. Please consult MSDN documentation for WDDM 1.1 requirements.

Send comments about this topic to Microsoft

Device.Graphics.WDDM11.Display

The base feature set implemented by drivers supporting all versions of the WDDM Display DDIs.

In this topic:

Device.Graphics.WDDM11.Display.Base

A graphic drivers that is written for a discrete graphic adapter or an integrated graphics adapter device must meet all requirements defined in the WDDM 1.1 specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

See requirement **Device.Graphics.WDDM11.Base**

Send comments about this topic to Microsoft

Device.Graphics.WDDM11.DisplayRender

The base feature set implemented by drivers supporting all versions of the WDDM for both Display and Render DDIs.

In this topic:

Device.Graphics.WDDM11.DisplayRender.Base

A graphic driver that is written for a discrete graphic adapter or an integrated graphics adapter device must meet all requirements defined in the WDDM 1.1 specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

See requirement **Device.Graphics.WDDM11.Base**

Send comments about this topic to Microsoft

Device.Graphics.WDDM11.DisplayRender.D3D9Overlay

The optional feature implemented by WDDM 1.1 drivers and greater allowing for surfaces to be presented in a hardware overlay.

In this topic:

Device.Graphics.WDDM11.DisplayRender.D3D9Overlay.D3D9Overlay

A WDDM1.1 driver must support Direct3D 9 overlays.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the WDDM1.1 driver supports Direct3D 9 overlays, the video overlay presentation requirements are as follows:

- A WDDM v1.1 driver must set the D3DCAPS_OVERLAY bit in the D3DCaps9.Caps field.
- A WDDM v1.1 driver must support query type D3DDDICAPS_CHECKOVERLAYSUPPORT to the user mode pfnGetCaps DDI call.
- A WDDM v1.1 driver must support overlays in at least one valid configuration (Displaymode, OverlayFormat, Width, and Height) when called to DDICHECKOVERLAYSUPPORTDATA for supported overlay and the Max width and height of supported overlay must be greater than zero.

Send comments about this topic to Microsoft

Device.Graphics.WDDM11.Render

The base feature set implemented by drivers supporting all versions of the WDDM Render DDIs.

In this topic:

Device.Graphics.WDDM11.Render.Base

A graphic driver that is written for a discrete graphic adapter or an integrated graphics adapter device must meet all requirements defined in the WDDM 1.1 specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

See requirement **Device.Graphics.WDDM11.Base**

Send comments about this topic to Microsoft

Device.Graphics.WDDM11.Render.DXVAHD

The optional feature that is implemented by WDDM 1.1 drivers supporting the new state-based video processing DDIs.

In this topic:

Device.Graphics.WDDM11.Render.DXVAHD.DXVAHD

WDDM1.1 driver supports DXVA-HD

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the WDDM1.1 driver supports DXVA-HD, then the following input formats (D3DDDICAPS_DXVAHD_GETVPINPUTFORMATS) must be supported:

- YUY2
- AYUV
- NV12
- X8R8G8B8

- A8R8G8B8

Also the driver must support the following output formats (D3DDDICAPS_DXVAHD_GETVPOUTPUTFORMATS):

- X8R8G8B8
- A8R8G8B8

Send comments about this topic to Microsoft

Device.Graphics.WDDM12

The base feature set implemented by drivers supporting WDDM 1.2.

In this topic:

Device.Graphics.WDDM12.Base

Graphics drivers must be implemented per the WDDM 1.2 specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The WDDM v1.2 is a superset of WDDM 1.1, and WDDM 1.0.

Below is a summary these WDDM versions:

Operating System	Driver Models Supported	D3D versions supported	Features enabled
Windows Vista	WDDM 1.0 XDDM on Server and limited UMPC	D3D9, D3D10	Scheduling, Memory Management, Fault tolerance, D3D9 & 10
Windows Vista SP1 / Windows 7 client pack	WDDM 1.05 XDDM on Server 2008	D3D9, D3D10, D3D10.1	+ BGRA support in D3D10, D3D 10.1

Windows 7	WDDM 1.1 XDDM on Server 2008 R2	D3D9, D3D10, D3D10.1, D3D11	GDI Hardware acceleration, Connecting and configuring Displays, DXVA HD, D3D11
Windows 8	WDDM 1.2	D3D9, D3D10, D3D10.1, D3D11, D3D11.1	Smooth Rotation, 3D Stereo, D3D11 Video, GPU Preemption, TDR Improvements Diagnostic Improvements, Performance and Memory usage Optimizations, Power Management, etc.

WDDM v1.2 also introduces new types of graphics drivers, targeting specific scenarios and is described below:

- WDDM Full Graphics Driver: This is the full version of the WDDM graphics driver that supports hardware accelerated 2D & 3D operations. This driver is fully capable of handling all the render, display, and video functions. WDDM 1.0 and WDDM 1.1 are full graphics drivers. All Windows 8 client systems must have a full graphics WDDM 1.2 device as the primary boot device.
- WDDM Display Only Driver: This driver is only supported as a WDDM 1.2 driver and enables IHVs to write a WDDM based kernel mode driver that is capable of driving display only devices. The OS handles the 2D or 3D rendering using a software simulated GPU.
- WDDM Render Only Driver: This driver is only supported as a WDDM 1.2 driver and enables IHVs to write a WDDM driver that supports only rendering functionality. Render only devices are not allowed as the primary graphics device on client systems.

Table below explains the scenario usage for the new driver types:

	Client	Server	Client running in a Virtual Environment	Server Virtual
Full Graphics	Required as boot device	Optional	Optional	Optional

Display Only	Not allowed	Optional	Optional	Optional
Render Only	Optional as non primary adapter	Optional	Optional	Optional
Headless	Not allowed	Optional	N/A	N/A

WDDM v1.2 Feature caps

The table below lists the requirements for a WDDM v1.2 driver to specify to Windows the WDDM Driver Type, version, and the feature caps (visible to dxgkrnl) that WDDM v1.2 drivers are required to set. If a driver has wrongfully claimed itself as "WDDM v1.2" or has implemented partial features (only some of the mandatory features), then it will fail to create an adapter and the system will fall back to the Microsoft Basic Display Driver.

WDDM Driver Requirements

WDDM driver type	DDI requirements
Full Graphics	Implement all the Render-specific and the Display-specific required DDIs
Display-Only	Implement all the Display-specific DDIs and return a null pointer for all the Render-specific DDIs
Render-Only	Implement all the Render-specific DDIs and return a null pointer for all the Display-specific DDIs **OR** Implement all the DDIs for a full WDDM driver but report DISPLAY_ADAPTER_INFO::NumVidPnSources = 0 and DISPLAY_ADAPTER_INFO::NumVidPnTargets = 0

WDDM v1.2 Feature Caps

Feature	WDDM Driver Type			Feature Caps

	Full Graphics	Render Only	Display Only	
WDDM version	**M**	**M**	**M**	DXGK_DRIVERCAPS::WDDMVersion
Bugcheck and PnP Stop support for Non VGA	**M**	**NA**	**M**	DXGK_DRIVERCAPS::SupportNonVGA
Optimized screen rotation Support	**M**	**NA**	**M**	DXGK_DRIVERCAPS::SupportSmoothRotation
GPU Preemption	**M**	**M**	**NA**	DXGK_DRIVERCAPS::PreemptionCaps
FlipOnVSyncMmIo	**M**	**M**	**NA**	DXGK_FLIPCAPS::FlipOnVSyncMmIo FlipOnVSyncMmIo is NOT a new feature. This feature is already documented and has been available since Windows Vista; the requirement here is to set the FlipOnVSyncMmIo cap.
TDR Improvements	**M**	**M**	**NA**	DXGK_DRIVERCAPS::SupportPerEngineTDR
Optimizing the graphics stack to improve performance on sleep & resume	**O**	**O**	**NA**	DXGK_SEGMENTDESCRIPTOR3::Flags
Stereoscopic 3D: New infrastructure to process and present stereoscopic content	**O**	**NA**	**NA**	D3DKMDT_VIDPN_SOURCE_MODE_TYPE
DirectFlip	**M**	**NA**	**NA**	DXGK_DRIVERCAPS::SupportDirectFlip

GDI Hardware acceleration (This is a required WDDM v1.1 feature)	**M**	**M**	**NA**	DXGK_PRESENTATIONCAPS::SupportKernelModeCommandBuffer
GPU power management of idle and active power	**O**	**O**	**O**	If this feature is supported, the DDI functions must be supported (SetPowerComponentFState and PowerRuntimeControlRequest).

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.Display

Display feature requirements for all WDDM12 drivers that support the display specific DDIs

In this topic:

Device.Graphics.WDDM12.Display.Base

Requirements for a WDDM graphics adapter to support display functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM has been extended to support a WDDM driver that is only responsible for Display Scan out capabilities. Such a driver is not allowed to support any Rendering capabilities.

A driver is considered a WDDM Display Only driver if it implements the following DDIs.

Common WDDM DDIs

These DDIs are for the common device functionalities such as PnP support and Power support. These functions are required by all WDDM drivers, and if not implemented, the driver will not be started by Windows. These DDIs are already documented in the WDK.

Required:

DxgkDdiAddDevice

DxgkDdiStartDevice

DxgkDdiStopDevice

DxgkDdiRemoveDevice

DxgkDdiDispatchIoRequest

DxgkDdiSetPowerState

DxgkDdiUnload

Optional:

DxgkDdiInterruptRoutine*

DxgkDdiDpcRoutine

DxgkDdiNotifyAcpiEvent

DxgkDdiQueryInterface

DxgkDdiControlEtwLogging

DxgkDdiEscape

DxgkDdiCollectDbgInfo

*DxgkDdiInterruptRoutine function is required if the current hardware device reports a hardware interrupt. In this case, if the driver did not supply this DDI function, the OS would fail the initialization. If the current hardware does not have a hardware interrupt and the driver supplies this DDI function, the OS will still allow this driver to be loaded and DxgkDdiInterruptRoutine will never be called.

* DdiNotifyAcpiEvent DDI function is used to notify graphics drivers on some ACPI events. It is optional for the rendering device. On normal WDDM graphics devices, this DDI function will return a flag to indicate dxgkrnl.sys to take some further actions such as reset display mode or poll the connected monitors. For the rendering only device, these flags are not used and must be set to zero.

Display Only DDIs

The following DDI functions are required to be implemented by a Display Only driver. If the driver does not supply all of these DDIs, Windows will fail to initialize this driver.

DxgkDdiQueryChildRelations

DxgkDdiQueryChildStatus

DxgkDdiQueryDeviceDescriptor

DxgkDdiResetDevice

DxgkDdiQueryAdapterInfo

DxgkDdiSetPalette *

DxgkDdiSetPointerPosition **

DxgkDdiSetPointerShape **

DxgkDdiIsSupportedVidPn

DxgkDdiRecommendFunctionalVidPn ***

DxgkDdiEnumVidPnCofuncModality

DxgkDdiSetVidPnSourceVisibility

DxgkDdiCommitVidPn

DxgkDdiUpdateActiveVidPnPresentPath

DxgkDdiRecommendMonitorModes

DxgkDdiGetScanLine

DxgkDdiQueryVidPnHWCapability

DxgkDdiPresentDisplayOnly

DxgkDdiReleasePostDisplayOwnership

DxgkDdiSystemDisplayEnable

DxgkDdiSystemDisplayWrite

DxgkDdiI2CReceiveDataFromDisplay

DxgkDdiI2CTransmitDataFromDisplay

*DxgkDdiSetPalette is a required DDI. If the driver does not support any palette mode, it should still supply this DDI.

**DxgkDdiSetPointerPosition and DxgkDdiSetPointerShape are required DDIs. If the driver does not support any hardware cursor, it should still supply these two DDIs.

***DxgkDdiRecommendFunctionalVidPn is a required DDI. If the driver does not support any ACPI event, it should still supply this DDI.

Device.Graphics.WDDM12.Display.ContainerIDSupport

A graphics adapter must support the DDI for Container ID.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A graphics adapter must implement the **DxgkDdiGetDeviceContainer** DDI. Windows will use this DDI to ask the driver for the container ID. The driver must try and obtain the Container ID from the display hardware.

In case the Display device does not provide a Container ID, then Windows will automatically manufacture a Container ID for the display. The uniqueness of the Container ID is achieved by using the Manufacture ID, Product ID, and Serial number of the display obtained from the EDID. Using this information, a driver for another device can generate the same container ID as the display device by using the RtlGenerateClass5Guid function included in wdm.h.

Device.Graphics.WDDM12.Display.DisplayOutputControl

Support for WDDM taking control of display output while the WDDM driver is running

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

There is a need for a more seamless hand off of the display control between Windows and the WDDM graphics driver. This means that in some cases, Windows needs to take over the control of the Display scan out without having to PnP Stop the WDDM Driver.

One such scenario is when Windows needs to bug check the system and display the blue screen. This set of DDIs enables that cleaner hand off.

The following DDIs are required to be implemented by a Full and Display Only WDDM 1.2 driver.

DxgkDdiSystemDisplayEnable

DxgkDdiSystemDisplayWrite

These DDIs are documented here on WDK: http://msdn.microsoft.com/en-us/library/ff554066(v=VS.85).aspx.

The following are the requirements for WDDM driver while implementing these DDIs:

- When Windows calls DxgkDdiSystemDisplayEnable, a driver must cancel all the GPU operations or reset GPU to an idle state.
- Windows will provide a Target ID as part of the DDI call. The driver must continue to drive the display associated with the target ID.
    - The driver must check the connectivity of the display on the provided Target ID. If specified target does not have a display connected, the driver should fail this call with STATUS_NOT_SUPPORTED.

- The driver must disable the signal to all other displays connected to the adapter except the target ID provided.
    - If this is not possible, the driver should try and put a blank image on all other displays.
    - If this is not possible, the driver must leave the last image on the screen unchanged.
- For the selected Target ID, the driver must keep the current display mode and provide this mode back to Windows as part of the DDI call.
    - In case the driver is not able to maintain the current mode OR the target ID is not part of the active topology, the driver should try and set the native resolution of the display OR a high res mode. At the very least, the driver must reset to a mode that is larger than or equals to 640x480 24 bpp color format on the specified target.
    - It is NOT required that the driver should use linear frame buffer mode. But the driver should support the write operation from a D3DDDIFMT_A8R8G8B8 source to this frame buffer.
- This function might be called at the high IRQL level or after the system has been bugcheck. The driver should put this function in the non-paged code section and only use non-paged memory.
    - Windows kernel mode functions might NOT be available when this function is called.
- Once the driver has handed over Display Control to Windows, Windows will use the DxgkDdiSystemDisplayWrite DDI to update the screen image. Windows will use the DDI to write a block of image from specified source image to the screen that is reset via DxgkDdiSystemDisplayEnable function. This function will pass to the driver the start address of source image as well as the stride, width, and height. The color format of source image is always D3DDDIFMT_X8R8G8B8. Windows guarantees that the source image is in non-paged memory. The driver must write this source image to current screen at (PostionX, PositionY).
- This function might be called at the high IRQL level or after the system has been bugcheck. The driver should put this function in the non-paged code section and only use non-paged memory.
    - Windows kernel mode functions might NOT be available when this function is called.
- It is recommended to use the CPU to write the image from source to the frame buffer since the bugcheck might be caused by repeated TDR and the GPU might be in an unknown condition.

Device.Graphics.WDDM12.Display.ModeEnumeration

Mode enumeration requirements

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Target modes

- A graphics adapter must be able to scan out any resolution that is greater than or equal to 1024 x 768 progressive at 32 bpp and a maximum timing that is up to 148.5 Mhz pixel clock per the VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT) Nov 2007 Update or CEA-861-E specs. The graphics driver may support modes that are greater than or less than these constraints, but the driver is not required to do so.
- The graphics drier is not required to support any interlaced timings, but may choose to do so.
- During DxgkDdiEnumVidPnCofuncModality DDI call, the supported target modes must be greater than or equal to the pinned source modes except for analog TV connector type or if the target cannot support any timing with a resolution of 1024 x 768 or greater. This means that for all other conditions, the driver is only allowed to scale up. A driver can support downscaling if the user requests it specifically for overscan compensation on TVs.

Source modes

- A graphics adapter must support the native resolution of the integrated panel.
- If the graphics adapter has enough bandwidth, it must support the native resolution of any connected display.
- A graphics driver must enumerate at least one source mode for every achievable detailed timing in the EDID of the display.
- If the only mode supported by the display device is less than or equal to 640 x 480, then the driver can downscale in this case. However, the source mode must be at least 800 x 600.
- A graphics driver must only enumerate sources modes of 32 bpp or higher.

For Duplicate (Clone) mode

- If there are 2 displays configured in duplicate mode, the graphics adapter must support the setting of the following configuration:

- If there are any common detailed timings between the 2 displays that are less than or equal to 1920 x 1080 progressive at 32 bpp, the graphics adapter must support driving the displays at this timing in the duplicate mode.

- At a minimum, 1024 x 768 must be supported in duplicate mode.
- For more than 2 displays configured in duplicate mode, the graphics adapter may choose an appropriate mode to support.
- At a minimum, irrespective of the number of displays in duplicate mode, the graphics adapter must be able to support a Source Mode of 1024 x 768 progressive at 32 bpp in the duplicate mode.

For Extend mode

- If there are 2 displays configured in extended mode, the graphics adapter must support setting the following configuration:
  - On systems with integrated displays: Native resolution on integrated display and simultaneously up to 1920 x 1080 progressive at 32 bpp on the non-integrated display
  - Up to 1920 x 1080 progressive at 32 bpp on each non-integrated display
- For more than 2 displays configured in extended mode, the graphics adapter may choose an appropriate set of modes to support based on available bandwidth.

Device.Graphics.WDDM12.Display.PnpStopStartSupport

Support for PnP Stop in WDDM

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Since the introduction of WDDM, every driver is required to support PnP Start and Stop. This requirement enhances the support for PnP start and stop in WDDM.

Once a WDDM driver is stopped, Windows needs to take over the display control and when the driver is started, Windows needs to hand back display control to the driver.

WDDM 1.2 introduces a new DDI to add support for UEFI based systems and also to improve the user experience in such a scenario.

The following DDIs are required to be implemented by a Full and Display Only WDDM 1.2 driver.

DxgkDdiReleasePostDisplayOwnership

The following are the requirements for the driver when implementing this DDI.

- Windows will provide a Target ID as part of the DDI call. The driver must continue to drive the display associated with the target ID.
    - The driver must check the connectivity of the display on the provided Target ID. If specified target does not have a display connected, the driver should fail this call with STATUS_NOT_SUPPORTED.
    - The driver must disable the signal to all other displays connected to the adapter except the target ID provided.
        - If this is not possible, the driver should try and put a blank image on all other displays.
        - If this is not possible, the driver must leave the last image on the screen unchanged.
- If there is an ACPI ID associated with the target ID, then that must be provided back to Windows as part of the return data structure PDXGK_DISPLAY_INFORMATION.
- For the selected Target ID, the driver must keep the current display mode and provide this mode back to Windows as part of the DDI call.
    - In case the driver is not able to maintain the current mode OR the target ID is not part of the active topology, the driver should select an alternate active target and try and maintain the current resolution of that target. If that is not possible, the driver should try and set the native resolution of the display OR a high res mode. At the very least, the driver must reset to a mode which is larger than or equals to 800x600 24bpp (D3DDDIFMT_R8G8B8) or 32bpp (D3DDDIFMT_X8R8G8B8).
    - If no target is active, then the driver should attempt to enable a target. Preferably the internal panel (if available).
- The driver must clean the current frame buffer, disable hardware cursor, disable all overlays, and set to default GAMMA ramp.
- The driver must make the current frame buffer linear (either by using swizzle range or disabling swizzle mode).
- The driver must make the current frame buffer accessible by CPU.
- The driver must ensure that visibility is set to "enabled" on the specified target.

Device.Graphics.WDDM12.Display.ProvideLinearFrameBuffer

A graphics device can provide a linear frame buffer usable by Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

There are numerous scenarios where components in the Windows OS need to be able to update the display when an IHV driver is not loaded. Some of these scenarios are: Boot, Bugcheck, safemode, driver upgrades, etc. To ensure a graphics device is compatible with these Windows scenarios, it must:

- Ensure updates to the frame buffer must be scanned out without requiring addition IHV/OEM.
- software/drivers. Provide a linear frame buffer that is CPU accessible on demand from the IHV driver.
- On UEFI systems at boot using UEFI 2.3 GOP.
- On BIOS systems using VBE 3.0 standards.

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.DisplayOnly

The optional feature set implemented by WDDM 1.2 drivers that support only the display specific DDIs.

In this topic:

Device.Graphics.WDDM12.DisplayOnly.Base

Requirements for a WDDM graphics adapter to support display functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM has been extended to support a WDDM driver that is only responsible for Display Scan out capabilities. Such a driver is not allowed to support any Rendering capabilities.

A driver is considered a WDDM Display Only driver if it only implements the following DDIs and does not implement any of the render DDIs.

Common WDDM DDIs

These DDIs are for the common device functionalities such as PnP support and Power support. These functions are required by all WDDM drivers, and if not implemented, the driver will not be started by Windows. These DDIs are already documented in the WDK: http://msdn.microsoft.com/en-us/library/ff554066(v=VS.85).aspx.

Required:

DxgkDdiAddDevice

DxgkDdiStartDevice

DxgkDdiStopDevice

DxgkDdiRemoveDevice

DxgkDdiDispatchIoRequest

DxgkDdiSetPowerState

DxgkDdiUnload

Optional:

DxgkDdiInterruptRoutine*

DxgkDdiDpcRoutine

DxgkDdiNotifyAcpiEvent

DxgkDdiQueryInterface

DxgkDdiControlEtwLogging

DxgkDdiEscape

DxgkDdiCollectDbgInfo

*DxgkDdiInterruptRoutine function is required if the current hardware device reports hardware interrupt. In this case, if the driver did not supply this DDI function, the OS would fail the initialization. If the current hardware does not have interrupt and the driver supplies this DDI function, the OS will still allow this driver to be loaded and DxgkDdiInterruptRoutine will never be called.

* DdiNotifyAcpiEvent DDI function is used to notify graphics drivers on some ACPI events. It is optional for a rendering device. On normal WDDM graphics devices, this DDI function will return a flag to indicate dxgkrnl.sys to take some further actions such as reset display mode or poll the connected monitors. For the rendering only device, these flags are not used and must be set to zero.

Display Only DDIs

Following DDI functions are required to be implemented by a Display Only driver. If the driver does not supply all of these DDIs, Windows will fail to initialize this driver.

DxgkDdiQueryChildRelations

DxgkDdiQueryChildStatus

DxgkDdiQueryDeviceDescriptor

DxgkDdiResetDevice

DxgkDdiQueryAdapterInfo

DxgkDdiSetPalette *

DxgkDdiSetPointerPosition **

DxgkDdiSetPointerShape **

DxgkDdiIsSupportedVidPn

DxgkDdiRecommendFunctionalVidPn ***

DxgkDdiEnumVidPnCofuncModality

DxgkDdiSetVidPnSourceVisibility

DxgkDdiCommitVidPn

DxgkDdiUpdateActiveVidPnPresentPath

DxgkDdiRecommendMonitorModes

DxgkDdiGetScanLine

DxgkDdiQueryVidPnHWCapability

DxgkDdiPresentDisplayOnly

DxgkDdiReleasePostDisplayOwnership

DxgkDdiSystemDisplayEnable

DxgkDdiSystemDisplayWrite

*DxgkDdiSetPalette is a required DDI. If the driver does not support any palette mode, it should still supply this DDI.

**DxgkDdiSetPointerPosition and DxgkDdiSetPointerShape are required DDIs. If the driver does not support any hardware cursor, it should still supply these two DDIs.

***DxgkDdiRecommendFunctionalVidPn is a required DDI. If the driver does not support any ACPI event, it should still supply this DDI.

Design Note:

The only color format supported for display only drivers is D3DDDIFMT_A8R8G8B8; therefore, these drivers should only enumerate source modes of this format.

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.DisplayRender

The optional feature set implemented by WDDM 1.2 drivers supporting both display and render DDIs.

In this topic:

Device.Graphics.WDDM12.DisplayRender.Base

Requirements for a WDDM graphics adapter implementing both Render and Display DDIs

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM has been extended to support a WDDM driver that is only responsible for Display Scan out capabilities. Such a driver is not allowed to support any Rendering capabilities.

WDDM has also been extended to support a WDDM driver that is only responsible for Rendering and Compute DDIs. Such a driver is not allowed to support any Display Scan out capabilities.

Driver's implementing both sets of DDI's (Display and Render) are considered full graphics adapters.

Common WDDM DDIs

These DDIs are for the common device functionalities such as PnP support and Power support. These functions are required by all WDDM drivers, and if not implemented, the driver will not be started by Windows. These DDIs are already documented in the WDK: http://msdn.microsoft.com/en-us/library/ff554066(v=VS.85).aspx.

Required:

DxgkDdiAddDevice

DxgkDdiStartDevice

DxgkDdiStopDevice

DxgkDdiRemoveDevice

DxgkDdiDispatchIoRequest

DxgkDdiSetPowerState

DxgkDdiUnload

Optional:

DxgkDdiInterruptRoutine*

DxgkDdiDpcRoutine

DxgkDdiNotifyAcpiEvent

DxgkDdiQueryInterface

DxgkDdiControlEtwLogging

DxgkDdiEscape

DxgkDdiCollectDbgInfo

*DxgkDdiInterruptRoutine function is required if current hardware device reports hardware interrupt. In this case, if the driver did not supply this DDI function, the OS would fail the initialization. If the current hardware does not have an interrupt and the driver supplies this DDI function, the OS will still allow this driver to be loaded and DxgkDdiInterruptRoutine will never be called.

Display Only DDIs

Following DDI functions are required to be implemented by a Display Only driver. If the driver does not supply all of these DDIs, Windows will fail to initialize this driver.

DxgkDdiQueryChildRelations

DxgkDdiQueryChildStatus

DxgkDdiQueryDeviceDescriptor

DxgkDdiResetDevice

DxgkDdiQueryAdapterInfo

DxgkDdiSetPalette *

DxgkDdiSetPointerPosition **

DxgkDdiSetPointerShape **

DxgkDdiIsSupportedVidPn

DxgkDdiRecommendFunctionalVidPn ***

DxgkDdiEnumVidPnCofuncModality

DxgkDdiSetVidPnSourceVisibility

DxgkDdiCommitVidPn

DxgkDdiUpdateActiveVidPnPresentPath

DxgkDdiRecommendMonitorModes

DxgkDdiGetScanLine

DxgkDdiQueryVidPnHWCapability

DxgkDdiPresentDisplayOnly

DxgkDdiReleasePostDisplayOwnership

DxgkDdiSystemDisplayEnable

DxgkDdiSystemDisplayWrite

*DxgkDdiSetPalette is a required DDI. If the driver does not support any palette mode, it should still supply this DDI.

**DxgkDdiSetPointerPosition and DxgkDdiSetPointerShape are required DDIs. If the driver does not support any hardware cursor, it should still supply these two DDIs.

***DxgkDdiRecommendFunctionalVidPn is a required DDI. If the driver does not support any ACPI event, it should still supply this DDI.

* DdiNotifyAcpiEvent DDI function is used to notify graphics drivers on some ACPI events. It is optional for a rendering device. On normal WDDM graphics devices, this DDI function will return a flag to indicate dxgkrnl.sys to take some further actions such as reset the display mode or poll the connected monitors. For the rendering only device, these flags are not used and must be set to zero.

Rendering only DDIs

These DDI functions are for the rendering functionalities.

Required:

DxgkDdiInterruptRoutine

DxgkDdiDpcRoutine

DxgkDdiResetDevice

DxgkDdiCreateDevice

DxgkDdiDestroyAllocation

DxgkDdiDescribeAllocation

DxgkDdiOpenAllocation

DxgkDdiCloseAllocation

DxgkDdiGetStandardAllocationDriverData

DxgkDdiSubmitCommand

DxgkDdiPreemptCommand

DxgkDdiBuildPagingBuffer

DxgkDdiResetFromTimeout

DxgkDdiRestartFromTimeout

DxgkDdiQueryCurrentFence

DxgkDdiControlInterrupt

DxgkDdiDestroyDevice

DxgkDdiPresent

DxgkDdiCreateAllocation

DxgkDdiPatch

DxgkDdiRender

DxgkDdiRenderKm

Optional:

If the rendering hardware supports swizzling ranger, the driver should also supply the two following functions:

DxgkDdiAcquireSwizzlingRange

DxgkDdiReleaseSwizzlingRange

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.DisplayRender.ProcessingStereoscopicVideoContent

The optional feature set implemented by WDDM 1.2 drivers that support stereoscopic video processing.

In this topic:

Device.Graphics.WDDM12.DisplayRender.ProcessingStereoscopicVideoContent.ProcessingStereoscopicVideoContent

If a display adapter supports presentation and processing of stereoscopic video content, it must conform to the DXGI specification for Stereo 3D.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM 1.2 drivers may optionally process video data encoded for stereo. If the driver publishes that it can support stereo, it must support the processing of content in horizontally and vertically arranged left and right eye views, as well as separate streams for the left and right eye views.

In addition, if the driver publishes that it can support stereo, it may optionally also process the following stereo content layouts:

- Row interleaved

- Column interleaved
- Checkerboard
- Flipped Configurations

Finally, the driver may optionally support new stereo processing features:

- Mono offset
- Stereo Eye Adjustment

For all optional content layouts and processing features, the driver must publish the appropriate cap to indicate if it supports the feature.

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.DisplayRender.RuntimePowerMgmt

The optional feature set implemented by WDDM 1.2 drivers that support the runtime power management DDIs.

In this topic:

Device.Graphics.WDDM12.DisplayRender.RuntimePowerMgmt.RuntimePowerMgmt

If implemented - WDDM 1.2 drivers must use the new WDDM 1.2 GPU Power Management DDI for F-state and p-state power management.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM 1.2 drivers can optionally support F-state and p-state Power Management.

For more details on the use of the SoC GPU Power Management WDDM v1.2 DDI, please refer to the Windows WDK documentation.

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.Render

The base feature set implemented by WDDM 1.2 drivers that support the render specific DDIs.

In this topic:

Device.Graphics.WDDM12.Render.Base

Requirements for a WDDM graphics adapter to support render functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM has been extended to support a WDDM driver that is only responsible for Rendering and Compute DDIs. Such a driver is not allowed to support any Display Scan out capabilities.

A driver is considered a WDDM Render Only driver if one of the following two conditions is met:

- The driver implements all the DDIs required for a full WDDM driver, but reports 0 sources and 0 targets.
- The driver implements all the Render specific DDIs and returns a null pointer for all the Display specific DDIs. The DDIs required for this are called out below.

Common WDDM DDIs

These DDI functions are for the common device functionalities such as PnP support and Power support. These functions are required by all WDDM drivers, and if not implemented, the driver will

not be started by Windows. These DDIs are already documented in the WDK: http://msdn.microsoft.com/en-us/library/ff554066(v=VS.85).aspx.

Required:

DxgkDdiAddDevice

DxgkDdiStartDevice

DxgkDdiStopDevice

DxgkDdiRemoveDevice

DxgkDdiDispatchIoRequest

DxgkDdiSetPowerState

DxgkDdiUnload

Optional:

DdiNotifyAcpiEvent *

* DdiNotifyAcpiEvent DDI function is used to notify graphics drivers on some ACPI events. It is optional for a rendering device. On normal WDDM graphics devices, this DDI function will return a flag to indicate dxgkrnl.sys to take some further actions such as reset display mode or poll the connected monitors. For the rendering only device, these flags are not used and must be set to zero.

Rendering only DDIs

These DDI functions are for the rendering functionalities.

Required:

DxgkDdiInterruptRoutine

DxgkDdiDpcRoutine

DxgkDdiResetDevice

DxgkDdiCreateDevice

DxgkDdiDestroyAllocation

DxgkDdiDescribeAllocation

DxgkDdiOpenAllocation

DxgkDdiCloseAllocation

DxgkDdiGetStandardAllocationDriverData

DxgkDdiSubmitCommand

DxgkDdiPreemptCommand

DxgkDdiBuildPagingBuffer

DxgkDdiResetFromTimeout

DxgkDdiRestartFromTimeout

DxgkDdiQueryCurrentFence

DxgkDdiControlInterrupt

DxgkDdiDestroyDevice

DxgkDdiPresent

DxgkDdiCreateAllocation

DxgkDdiPatch

DxgkDdiRender

DxgkDdiRenderKm

Optional:

If the rendering hardware supports swizzling ranger, the driver should also supply the two following functions:

DxgkDdiAcquireSwizzlingRange

DxgkDdiReleaseSwizzlingRange

Device.Graphics.WDDM12.Render.D3D11VideoDecoding

A display driver must support the DirectX 11 Video Decoder DDI.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Display WDDM 1.2 drivers support the DirectX 11 Video Decoder DDI.

WDDM drivers must support at least one of the following sets of H.264 GUIDs:

- D3D11_DECODER_PROFILE_H264_VLD_NOFGT
- D3D11_DECODER_PROFILE_H264_VLD_FGT
- D3D11_DECODER_PROFILE_H264_VLD_WITHFMOASO_NOFGT

WDDM drivers support at least one the following VC1 modes:

- D3D11_DECODER_PROFILE_VC1_VLD
- D3D11_DECODER_PROFILE_VC1_D2010

If-implemented: WDDM drivers support at least one of the following MPEG2 modes:

- D3D11_DECODER_PROFILE_MPEG2_VLD and D3D11_DECODER_PROFILE_MPEG2_IDCT
- D3D11_DECODER_PROFILE_MPEG2_VLD and D3D11_DECODER_PROFILE_MPEG2_MOCOMP

- D3D11_DECODER_PROFILE_MPEG2_IDCT
- D3D11_DECODER_PROFILE_MPEG2AND1_VLD

If-implemented: HEVC/H.265 modes

- D3D11_DECODER_PROFILE_HEVC_VLD_MAIN
- D3D11_DECODER_PROFILE_HEVC_VLD_MAIN10

It is highly recommended for drivers to support D3D11_DECODER_PROFILE_HEVC_VLD_MAIN10 as Main 10 is a very common format for premium HEVC content.

Finally, WDDM drivers must support DXGI_FORMAT_420_OPAQUE and DXGI_FORMAT_NV12 as decoder output formats. For HEVC Main 10, drivers must support DXGI_FORMAT_P010 as decoder output format.

WDDM drivers must support Standardized AES 128 (defined in the WDDM v1.1 specifications) for H.264*** and MPEG2, if MPEGE2 acceleration is supported.

Design Notes

*** Standardized AES 128 support is required on x86 and x64 architectures and operating systems only.

Device.Graphics.WDDM12.Render.D3D11VideoProcessing

A display driver must support the appropriate DDIs for DirectX 11 video processing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following video processor device capability DDIs must be supported:

- BOB Deinterlacing
  - DXVAHDDDI_PROCESSOR_CAPS_DEINTERLACE_BOB
  - D3D11_1DDI_VIDEO_PROCESSOR_PROCESSOR_CAPS_DEINTERLACE_BOB
- Constriction
  - DXVAHDDDI_FEATURE_CAPS_CONSTRICTION
  - D3D11_1DDI_VIDEO_PROCESSOR_FEATURE_CAPS_CONSTRICTION
- Rotation

    - DXVAHDDDI_FEATURE_CAPS_ROTATION
    - D3D11_1DDI_VIDEO_PROCESSOR_FEATURE_CAPS_ROTATION
- Arbitrary stretching/shrinking
- Both full and nominal RGB ranges must be supported on the input and the output YCbCr data, specifically:
    - 0 to 255
        - DXVAHDDDI_STREAM_STATE_INPUT_COLOR_SPACE_DATA

RGB_Range field is set to 0:

- DXVAHDDDI_BLT_STATE_OUTPUT_COLOR_SPACE_DATA RGB_Range field is set to 0
    - D3D11_1DDI_VIDEO_PROCESSOR_COLOR_SPACE RGB_Range field set to 0
- 16 to 235
    - DXVAHDDDI_STREAM_STATE_INPUT_COLOR_SPACE_DATA

RGB_Range field is set to 1:

- DXVAHDDDI_BLT_STATE_OUTPUT_COLOR_SPACE_DATA RGB_Range field is set to 1
    - D3D11_1DDI_VIDEO_PROCESSOR_COLOR_SPACE RGB_Range field set to 1
- BT. 601 and BT. 709 conversion matrices must be supported on the input and the output YCbCr data, specifically:
- DXVAHDDDI_STREAM_STATE_INPUT_COLOR_SPACE_DATA.YCbCr_Matrix
    - Both setting the field to 0 (representing BT.601) and 1 (representing BT.709) must be supported.
- DXVAHDDDI_BLT_STATE_OUTPUT_COLOR_SPACE_DATA.YCbCr_Matrix
    - Both setting the field to 0 (representing BT.601) and 1 (representing BT.709) must be supported.
- D3D11_1DDI_VIDEO_PROCESSOR_COLOR_SPACE.YCbCr_Matrix
    - Both setting the field to 0 (representing BT.601) and 1 (representing BT.709) must be supported.
- Arbitrary background color

- Per-stream source rectangle
- Per-stream destination rectangle
- Overall destination rectangle
- At least one stream

The following formats must be supported for input:

- D3DFMT_420_OPAQUE
- D3DFMT_NV12
- D3DFMT_YUY2
- Any formats supported as output from DXVA 2.0 decode or DirectX 11 Video decode

Depending on the feature level made available by the driver for Direct3D (e.g. 9.1 - 9.3 for Direct3D 9 DDIs, 10.0 for Direct3D 10 DDIs, etc.), the following formats must be supported for output:

- 9.1 - 9.3 Feature Levels
  - D3DFMT_A8R8G8B8
  - D3DFMT_NV12
- 10.0 - 10.1 Feature Levels
  - DXGI_FORMAT_R16G16B16A16_FLOAT
  - DXGI_FORMAT_R8G8B8A8_UNORM
  - DXGI_FORMAT_R10G10B10A2_UNORM
  - DXGI_FORMAT_R8G8B8A8_UNORM_SRGB
  - DXGI_FORMAT_NV12
  - The following formats are also required if the driver supports these as a swapchain format:
    - DXGI_FORMAT_B8G8R8A8_UNORM
    - DXGI_FORMAT_B8G8R8A8_UNORM_SRGB
    - DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM
- 11.0 Feature Level and beyond

- DXGI_FORMAT_R16G16B16A16_FLOAT
- DXGI_FORMAT_R8G8B8A8_UNORM
- DXGI_FORMAT_R10G10B10A2_UNORM
- DXGI_FORMAT_R8G8B8A8_UNORM_SRGB
- DXGI_FORMAT_NV12, DXGI_FORMAT_B8G8R8A8_UNORM
- DXGI_FORMAT_B8G8R8A8_UNORM_SRGB
- DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM

Design Notes

MPEG-2 support is required on x86 and x64 architectures and operating systems only.

Device.Graphics.WDDM12.Render.DirectFlip

A driver must support the DirectFlip DDI.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The driver must publish the SupportDirectFlip field as TRUE in the DXGK_DRIVERCAPS structure.

WDDM 1.2 drivers for the Direct3D 11 DDI must implement the D3D11_1DDI_CHECKDIRECTFLIPSUPPORT DDI and return TRUE from the DDI when it is called with at least the following parameters:

- The app resource matches the DWM resource in format and pixel dimensions.
- The D3DDDI_CHECKDIRECTFLIP_IMMEDIATE flag is not set.

WDDM 1.2 drivers for the Direct3D 9 DDI must implement the D3DDDIARG_CHECKDIRECTFLIPSUPPORT DDI and return TRUE from the DDI when it is called with at least the following parameters:

- The app resource matches the DWM resource in format and pixel dimensions.
- The D3DDDI_CHECKDIRECTFLIP_IMMEDIATE flag is not set.

The driver must support the creation of shared primary swapchains, specifically identified as resources created with:

- pPrimaryDesc as non-NULL.

- D3D10_DDI_RESOURCE_MISC_SHARED set in MiscFlags.
- D3D10_DDI_BIND_SHADER_RESOURCE, D3D10_DDI_BIND_PRESENT, and D3D10_DDI_BIND_RENDER_TARGET all set in BindFlags.

When the SharedPrimaryTransition bit is set in the DXGK_SETVIDVIDPNSOURCEADDRESS DDI, the driver must:

- Validate that the hardware can seamlessly switch between the two allocations.
- Perform a seamless switch to the new primary indicated by the DDI.

Device.Graphics.WDDM12.Render.FlipOnVSyncMmIo

WDDM1.2 drivers must support a memory mapped I/O (MMIO)-based flip that takes effect on the next vertical sync.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All WDDM1.2 drivers must publish the FlipOnVSyncMmIo field as TRUE in the DXGK_FLIPCAPS structure, and implement behavior outlined under FlipOnVSyncMmIo here:

http://msdn.microsoft.com/en-us/library/ff561069(v=VS.85).aspx.

Device.Graphics.WDDM12.Render.OfferReclaim

WDDM 1.2 drivers must use the Offer and Reclaim DDI to reduce the memory footprint.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM 1.2 drivers must use the new UMD Offer and Reclaim DDI to reduce the overhead of memory resources created for temporary surfaces in local video and system memory. An example is the use of staging buffers to accelerate the process of uploading data to GPU local memory. By offering these resources, the operating system can reuse the memory without risking the expense of destroying and recreating them at high frequency.

WDDM Drivers also often keep allocations that an application is no longer using so that if the application creates another allocation with the same properties, it can give back the old one. This

reduces the performance impact of rapidly destroying and re-creating allocations, a common bad behavior for applications, but it also can have a very high memory cost. By offering those allocations, the WDDM 1.2 driver can keep most of its performance without wasting memory.

Below are detailed sub-requirements:

- WDDM 1.2 drivers must always "Offer" internal staging buffers once they have been used. These temporary buffers generally hold data being moved or copied from a source to a destination. In Windows 7 and previous operating systems, these temporary surfaces are left in memory even though they were no longer in use.
- WDDM 1.2 drivers must always "Offer" the deferred deletions of surfaces that are recycled. If a driver decides to defer the destruction of a surface, it must at least offer it to the video memory manager.
- WDDM 1.2 driver should only Offer packed allocations when all of the individual resources contained in the allocation have been Offer-ed. The allocation as a whole should be reclaimed when any individual resources have been reclaimed.
- WDDM 1.2 drivers must always Offer discarded allocations if it implements its own renaming mechanism.
- WDDM 1.2 drivers should never pack allocations larger than 64 KB with other allocations, guaranteeing for application that offer will give them the expected benefit.

For more details on the use of the Offer and Reclaim WDDM v1.2 DDI, please refer to the Windows WDK documentation.

Device.Graphics.WDDM12.Render.PreemptionGranularity

WDDM 1.2 drivers must report the granularity level of GPU Preemption.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A WDDM 1.2 driver must report the GPU Preemption granularity for running Graphics and Compute scenarios.

The WDDM 1.2 must do the following:

- The WDDM 1.2 driver must first set the "PreemptionAware" flag and the "MultiEngineAware" flag in the _DXGK_VIDSCHCAPS structure through the DxgkDdiQueryAdapterInfo DDI.
- The WDDM 1.2 driver must set the appropriate GPU Preemption granularity through "D3DKMDT_PREEMPTION_CAPS PreemptionCaps".

There is no mandatory requirement on the actual level of preemption for Graphics or Compute. For example, if a GPU does not support any level of Mid-DMA Buffer Graphics Preemption such as that on the triangle boundary or others, then it can report this cap as D3DKMDT_GRAPHICS_PREEMPTION_DMA_BUFFER_BOUNDARY. Similarly, if it does not support any level of finer grained preemption for Compute, then it must report the cap as D3DKMDT_COMPUTE_PREEMPTION_DMA_BUFFER_BOUNDARY.

However, if the GPU does support Mid-DMA Buffer or Packet Preemption, then the WDDM 1.2 driver must choose the appropriate cap in order to take advantage of the benefits offered by finer grained GPU Preemption for Graphics and/or Compute.

For more details on the use of the GPU Preemption WDDM v1.2 DDI, please refer to the Windows WDK documentation.

Device.Graphics.WDDM12.Render.PremiumContentPlayback

Protected Environment Signature requirement for WDDM1.2 drivers

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Any WDDM 1.2 user mode graphics driver binary required for premium content playback must be properly signed as required by the Windows Protected Environment (PE) license program. Compliance with the PE license program is already required by section 3.1 of the PVP-OPM license agreement. Specifically, each binary in the required set must be signed to an approved root recognized by Windows OS Code Integrity for PE, and have either or both of the following signing characteristics:

- Catalog signature with an overall attribute of PE TRUSTED, with a hash entry for each specific file setting a field and value of PETrusted=1.
- Embedded signature with per-page hashes.

Additionally, the process of determining what signature each module needs is being standardized. Each INF file now must include a SignatureAttributes section uniquely identifying what type of signature is applicable for the associated driver binaries. Adding this section to existing inf files is a very simple process.

An example follows:

```
[SignatureAttributes]
NameOfFile.dll = SignatureAttributes.PETrust
[SignatureAttributes.PETrust]
PETrust=true
```

Device.Graphics.WDDM12.Render.TDRResiliency

WDDM 1.2 drivers for GPUs that support Per-Engine Reset must implement the Windows DDI for TDR Resiliency.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A WDDM 1.2 driver for a GPU supporting Per-Engine Reset must implement the TDR DDI for improved reliability and resiliency.

The WDDM 1.2 driver must do the following:

- The WDDM 1.2 driver must satisfy the WDDM 1.2 GPU Preemption requirement.
- The WDDM 1.2 driver must implement the following GPU Engine DDIs:
  - QueryDependentEngineGroup
  - QueryEngineStatus
  - ResetEngine
- WDDM 1.2 drivers must continue supporting the pre-Windows 8 TDR behavior of Adapter-wide Reset and Restart. Semantics of these existing DDIs remain the same as before Windows 8.

For more details on the use of the GPU Preemption and TDR Improvements WDDM v1.2 DDI, please refer to the Windows WDK documentation.

Device.Graphics.WDDM12.Render.UMDLogging

WDDM 1.2 drivers must implement WDDM User-Mode Driver or UMD Logging to aid diagnosability.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

A WDDM 1.2 driver must expose the relationship between the Direct3D resources and Video Memory allocations by implementing the UMD logging ETW interfaces. Below are the specific requirements for the drivers:

- UMD must report the allocation size specified in the CreateResource call, even if the size of the memory allocated is greater.
- UMD must log the MapAllocation event for each of its internal allocation and specify the purpose of that allocation.
- UMD must log the MapAllocation event for each renamed surface.
- UMD must log the MapAllocation for every surface that it packs into an existing surface.
- UMD must log the MapAllocation for every surface that currently exists in the event of a rundown.
- UMD must log the MapAllocation in response to CreateResource call, even if no new memory is allocated.
- UMD must log the UnmapAllocation each time its internal allocation is destroyed.
- UMD must log the UnmapAllocation each time an application destroys an allocation, even if actual memory allocation is not destroyed.
- UMD must log the UnmapAllocation each time it closes one of the renamed allocations.
- UMD must log the UnmapAllocation for each surface that is packed into a larger allocation.

In addition to logging MapAllocation and UnmapAllocation events as they happen, the Graphics Driver must be able to report all existing mappings between resources and allocations at any point in time.

For more details on the use of the UMD Logging WDDM v1.2 DDI, please refer to the Windows WDK documentation.

Device.Graphics.WDDM12.Render.XPSRasterizationConformance

WDDM 1.2 drivers must support XPS Rasterization.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

To ensure a quality printing experience on Windows, the WDDM1.2 graphic drivers must support XPS Rasterization.

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.RenderOnly

The optional feature set implemented by WDDM 1.2 drivers which support only the render specific DDIs.

In this topic:

Device.Graphics.WDDM12.RenderOnly.Base

Requirements for a WDDM graphics adapter to support render functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM has been extended to support a WDDM driver that is only responsible for Rendering and Compute DDIs. Such a driver is not allowed to support any Display Scan out capabilities.

A driver is considered a WDDM Render Only driver if one of the following two conditions is met:

- The driver implements all the DDIs required for a full WDDM driver, but reports 0 sources and 0 targets.
- The driver implements all the Render specific DDIs and returns a null pointer for all the Display specific DDIs. The DDIs required for this are called out below.

Common WDDM DDIs

These DDI functions are for the common device functionalities such as PnP support and Power support. These functions are required by all WDDM drivers, and if not implemented, the driver will not be started by Windows. These DDIs are already documented in the WDK, http://msdn.microsoft.com/en-us/library/ff554066(v=VS.85).aspx.

Required:

DxgkDdiAddDevice

DxgkDdiStartDevice

DxgkDdiStopDevice

DxgkDdiRemoveDevice

DxgkDdiDispatchIoRequest

DxgkDdiSetPowerState

DxgkDdiUnload

Optional:

DdiNotifyAcpiEvent *

* DdiNotifyAcpiEvent DDI function is used to notify graphics drivers on some ACPI events. It is optional for a rendering device. On normal WDDM graphics devices, this DDI function will return a flag to indicate dxgkrnl.sys to take some further actions such as reset display mode or poll the connected monitors. For the rendering only device, these flags are not used and must be set to zero.

Rendering only DDIs

These DDI functions are for the rendering functionalities.

Required:

DxgkDdiInterruptRoutine

DxgkDdiDpcRoutine

DxgkDdiResetDevice

DxgkDdiCreateDevice

DxgkDdiDestroyAllocation

DxgkDdiDescribeAllocation

DxgkDdiOpenAllocation

DxgkDdiCloseAllocation

DxgkDdiGetStandardAllocationDriverData

DxgkDdiSubmitCommand

DxgkDdiPreemptCommand

DxgkDdiBuildPagingBuffer

DxgkDdiResetFromTimeout

DxgkDdiRestartFromTimeout

DxgkDdiQueryCurrentFence

DxgkDdiControlInterrupt

DxgkDdiDestroyDevice

DxgkDdiPresent

DxgkDdiCreateAllocation

DxgkDdiPatch

DxgkDdiRender

DxgkDdiRenderKm

Optional:

If the rendering hardware supports swizzling ranger, the driver should also supply the two following functions:

DxgkDdiAcquireSwizzlingRange

DxgkDdiReleaseSwizzlingRange

Send comments about this topic to Microsoft

Device.Graphics.WDDM12.StandbyHibernateFlags

The optional feature implemented by WDDM 1.2 drivers supporting the new stand by hibernation flags.

In this topic:

Device.Graphics.WDDM12.StandbyHibernateFlags.StandbyHibernateFlags

WDDM v1.2 graphics drivers can optionally specify if they support the Windows optimized standby or hibernate flags.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To improve the performance on sleep & resume for standby and hibernate, a WDDM 1.2 driver can utilize the new StandbyHibernateFlags (PreservedDuringStandby, PreservedDuringHibernate, and PartiallyPreservedDuringHibernate). To utilize this feature, drivers must set one or more of the StandbyHibernateFlags when enumerating segment capabilities. Doing so indicates that eviction should be skipped during power state transition for the corresponding segments. Verification of memory retention during power transition will be verified for all WDDM 1.2 driver setting a StandbyHibernateFlag.

Send comments about this topic to Microsoft

Device.Graphics.WDDM13

The base feature set implemented by drivers supporting WDDM1.3.

In this topic:

Device.Graphics.WDDM13.Base

Graphic drivers must implement WDDM1.3.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM 1.3 is required for all new systems that are shipped with Windows 10.

Feature Name	Applicable DX H/W Class	Full Graphics Driver x86/x64	Render Only Driver x86/x64	Full Graphics Driver ARM/RT
Hybrid Graphics (GPU Kernel)	DX11+	If implemented	NA	NA
Multiplane Overlays (GPU Kernel)	All	If implemented	NA	Mandatory
Wireless display support	All	If implemented	NA	If implemented
48 Hz Video Playback	All	Mandatory	NA	Mandatory
Shared Surface support for more formats	All	Mandatory	Mandatory	Mandatory
Present Overhead Optimization	All	Mandatory	Mandatory	Mandatory

Kernel Performance: GPU engine enumeration	All	Mandatory	Mandatory	Mandatory
Power Management Enhancements: GPU p-state support	All	If implemented	If implemented	Mandatory
Power Management Enhancements: Stable P-State	All	Mandatory	Mandatory	Mandatory
Power Management Enhancements: GPU f-state hysteresis	All	If implemented*	If implemented	Mandatory
Power Management Enhancements: Aggressive V-sync	All	If implemented	NA	Mandatory
Access to thermal interface	All	If implemented	If implemented	If implemented
Rendering Performance Improvements: D3D9 Staging Buffers	All	Mandatory for DX9.x HW Optional for DX10+	NA	Mandatory
Rendering Performance Improvements: D3D9 Native UpdateSubResource	All	Mandatory for DX9.x HW Optional for DX10+	NA	Mandatory
Rendering Performance Improvements: D3D9 Timestamps and Counters	All	Mandatory for DX9.x HW Optional for DX10+	NA	Mandatory
Rendering Performance Improvements: D3D Resource Trim	All	Mandatory	Mandatory	Mandatory
Rendering Performance Improvements: Feature Level 9 Instancing	All	Mandatory for DX9.x HW	NA	Mandatory

Rendering Performance Improvements: D3D9 Large Capture Textures	All	Mandatory	NA	Mandatory
Rendering Performance Improvements: Map Default	DX11+	Mandatory	NA	NA
Tiled Resources	DX11.1	If implemented	If Implemented	If implemented
Independent Flip	All	Mandatory	NA	Mandatory
D3D Modification for Tools: High Performance Timing Data	All	Mandatory	Mandatory	Mandatory
Full Range YUV Support	DX10+	Mandatory	Mandatory	Mandatory

*****Mandatory if implementing Connected Standby

There are no additional requirements for Display only Driver; Hence, it is not called out in the table.

D3D Requirements:

DirectX Hardware	KMD Requirements	UMD Requirements
D3D9	Required: WDDM v1.3	Required: D3D9 - UMD DDI
D3D10	Required: WDDM v1.3	Required: D3D9 - UMD DDI Required: D3D10- UMD DDI Required: D3D11.1 - UMD DDI
D3D10.1	Required: WDDM v1.3	Required: D3D9 - UMD DDI Required: D3D10- UMD DDI Required: D3D10.1- UMD DDI Required: D3D11.1 - UMD DDI
D3D11	Required: WDDM v1.3	Required: D3D9 - UMD DDI Required: D3D10- UMD DDI Required: D3D10.1- UMD DDI Required: D3D11 - UMD DDI

		Required: D3D11.1 - UMD DDI
D3D11.1	Required: WDDM v1.3	Required: D3D9 - UMD DDI Required: D3D10- UMD DDI Required: D3D10.1- UMD DDI Required: D3D11 - UMD DDI Required: D3D11.1 - UMD DDI

Send comments about this topic to Microsoft

Device.Graphics.WDDM13.DisplayRender

The parent feature node of requirements that have an impact for the display and render GPU.

In this topic:

Device.Graphics.WDDM13.DisplayRender.48HzVideoPlayback

WDDM1.3 drivers must support 48 Hz Video Playback.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Graphic Drivers implementing WDDM1.3 must implement DDIs that:

- Report whether the collective system can seamlessly switch to 48 Hz (intersection of SOC and display panel capabilities).
- If so, allow the refresh rate (such as 48 Hz) to be specified on each particular Present DDI.

The test will validate that the refresh rate with within a 0.5% tolerance of the requested value, again assuming the graphics driver reports support for the particular refresh rate. This will be validated for devices thatreport support for 48 Hz or 50 Hz through the capabilities DDI through a test, which is described in detail in the 48 Hz IHV spec.

Device.Graphics.WDDM13.DisplayRender.IndependentFlip

IndependentFlip support

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All WDDM 1.3 drivers MUST support independentFlip. As such, all WDDM 1.3 drives MUST set the UINT FlipIndependent member to 1 in the updated DXGK_FLIPCAPS structure. See the IndependentFlip DDI Document for more details about the updated DXGK_FLIPCAPS structure.

Send comments about this topic to Microsoft

Device.Graphics.WDDM13.DisplayRender.CoolingInterface

Feature for Thermal Hints

In this topic:

Device.Graphics.WDDM13.DisplayRender.CoolingInterface.ThermalHints

Optional Thermal Hints support for WDDM1.3 drivers

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM graphics 1.3 or later drivers must check for the new added DeviceUid field in the QUERY_INTERFACE structure OS passed in by the DxgkDdiQueryInterface function and return a valid interface function on the OS specified device.

If ACPI firmware adds the current graphics device into ACPI ThermalZone, then the IHV miniport must support the DxgkDdiQueryInterface on the GUID_THERMAL_COOLING_INTERFACE for the current graphics device.

Validation:

ACPI.sys will only query the thermal cooling interface if the current device is in a thermal zone. Dxgkrnl.sys can validate that the IHV miniport driver supports the

GUID_THERMAL_COOLING_INTERFACE when ACPI queries this interface. In this case, WHLK can receive the error log that the miniport driver fails the DxgkDdiQueryInterface call on GUID_THERMAL_COOLING_INTERFACE and fail the WHLK test.

Send comments about this topic to Microsoft

Device.Graphics.WDDM13.DisplayRender.WirelessDisplay

The optional feature set implemented by WDDM 1.3 drivers that support the Miracast functionality

In this topic:

Device.Graphics.WDDM13.DisplayRender.WirelessDisplay.BasicWirelessDisplay

Optional Wireless Display support for WDDM1.3 drivers

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- This is an optional feature for WDDM 1.3 drivers.
- The WDDM Miracast user mode driver must be a separate DLL.
- If a driver supports Miracast display, then it must enumerate only one Miracast target along with other supported targets.
- A driver must accurately report the target type to Windows. Any time Windows connects to a Miracast device, the driver must use the new target type irrespective of the connection between the Miracast end point and the display device.
- A driver must only indicate the arrival of a monitor once the Miracast session is established, the device capabilities have been obtained, HDCP negotiations has taken place, and the user mode driver is ready to stream display to the device.
- A driver must use the new DDIs for any private communication between UMD and KMD. A driver must not poll for packet state.

- A driver must use the new DDIs for any private communication between UMD and KMD. A driver must not poll for packet state.
- All memory access by the GPU must be allocated through DirectX APIs. A driver must not carve out or hide memory that is used for the implementation.
- All hardware being used for the implementation must be exposed to Windows to enable accurate power management.
- A driver must report v-sync on the wireless displays.
- A driver must use published Windows networking APIs from the user mode and must not use any private interfaces with the NW driver.
- Every time the receiver requests an i-frame, the driver must log it with Windows using a new DDI.
- If the Miracast device supports Audio, the driver must enumerate an Audio end point to Windows similar to how it is enumerated for HDMI/Display Port. The container ID for the audio device must match that of the corresponding Miracast display device.
- When Windows sets the MC display to “monitor idle”, the driver must set the “power saving mode” as defined in the MC standard. When the user provides input to resume from monitor idle, the MC display must be brought back to full power and the user must be able to use it without having to reconnect.
- The user mode Miracast driver must complete the new DDI to stop a Miracast session, within 3 seconds.
- A driver must conform with the same resolution enumeration requirement as other target types as specified in the WHLK.
- In the event that there is no display physically connected to the Miracast sink, then the driver must still allow the connection to succeed but must not report the display to Windows. Once the display is physically connected, then the driver must report it to windows.
- In the event that a user physically disconnects the display from the Miracast sink device, the driver must report the display removal and Windows will disconnect from the session.
- A WDDM driver must be capable of supporting HDCP over Miracast.
- If a sink device does not support HDCP, the driver must still allow the connection to succeed.

- In case the driver TDR's, the driver should appropriately terminate the session.

Send comments about this topic to Microsoft

Device.Graphics.WDDM13.EnhancedPowerManagement

Graphics Kernel Power Management Enhancements

In this topic:

Device.Graphics.WDDM13.EnhancedPowerManagement.FState

F-State Hysteresis

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM 1.3 introduces some power management enhancements. If implemented then:

LATENCY TIME VALIDATION

For a transition from a lower power F-State to a higher one, it is expected that the time it takes to complete the transition will not be more than 10% over the transition latency time reported by the driver.

Device.Graphics.WDDM13.EnhancedPowerManagement.VSYNC

Aggressive VSYNC

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM 1.3 introduces some power management enhancements:

IN PHASE VSYNC

While the Vsync is enabled, 98% of the Vsyncs must be within 500 us of the expected value with no Vsyncs exceeding a variance of more than 1.5 ms.

Send comments about this topic to Microsoft

Device.Graphics.WDDM13.Render

The base feature set implemented by WDDM 1.3 drivers that support the render specific DDIs.

In this topic:

Device.Graphics.WDDM13.Render.CheckDDIBoundaries

Hybrid Graphics

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Graphics drivers should not circumvent DDI boundaries.

- No system level or runtime level API detouring is allowed – The driver is not allowed to intercept OS system or runtime calls, or modify arguments passed by OS system or runtime APIs, or wrap the objects returned from the API entrypoints.
- Access and use of internal data structures is not allowed.
- No driver layering is allowed - Only one user mode/kernel mode driver is allowed to be loaded and communicate with the runtime and kernel binaries.

Device.Graphics.WDDM13.Render.DirtyRect

Optional DirtyRect support for WDDM1.3 drivers

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM1.3 introduces the ability to support DirtyRect. If DirtyRect is implemented, then it must follow the specification.

Device.Graphics.WDDM13.Render.GPUNode

Graphics Kernel Performance

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If WDDM1.3 is implemented, then the driver must:

- The driver must specify a valid enum value for each engine per physical adapter.
- The driver must specify one engine of type DXGK_ENGINE_TYPE_3D per physical adapter.
- For each engine, the given enum value must meet the requirements as stated in the engine definitions table in the DDI.

Device.Graphics.WDDM13.Render.HighPerformanceTimingData

High Performance Graphics Timing Data

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

“High Performance Graphics Timing Data” provides lightweight and highly detailed timing data for graphics workloads. This data is used by analysis tools to identify performance or other graphics related issues in Windows applications or the graphics stack.

A WDDM1.3 driver must ensure that the graphics device/driver provides the following:

- A high resolution GPU Timer
  - 12.5 MHz (80ns resolution) or better.
  - At least 32 bits of timestamp resolution.
  - The GPU timestamp can be sampled for all engines in a GPU.
  - The GPU timestamp can be sampled at the end of the GPU pipeline.
  - The GPU timestamp frequency can be sampled.
  - The GPU timestamp is invariant and is unaffected by p-state transitions.
    - Among the other changes for this feature, this will include the addition of two new flags to the existing DdiControlEtwLogging interface; when these flags are set so that the first flag is value of 1 and the second flag is a value of 0, then the driver must ensure that:
      - All engine components must ensure that they are never clock- or power-gated as long as the flag remains enabled, and must in general refrain from entering any idle states. The components must remain active to ensure that there is no latency added due to power transitions.
      - All engine components must ensure that their processing frequencies and functional bus bandwidths are kept at their maximum stable operating values. Thermal events requiring P-State transition down should still occur to prevent damage to the hardware, but P-States should be defined so that these are exceptional occurrences that are not normally seen in cool lab environments.
- The sampled GPU timestamp can be correlated with previously issued graphics commands.
- The generation of timing data is off by default, but can be turned on and off at any time.
- The overhead of collecting this performance data is no slower than using the timestamp query technique that is already available in D3D9 and D3D10+.
- Timing data can be collected via the D3D9 driver on feature level 9 hardware, and the D3D10 driver on feature level 10+ hardware.
- The resulting data on Tile based deferred rendering hardware appears as if it was generated on an immediate mode rendering device.

Device.Graphics.WDDM13.Render.PresentOverheadOptimization

Present Overhead Optimization

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM1.3 drivers must support the new DDI, Present1, on Feature Level 9 hardware through the D3D9 DDI, as well as on Feature 10+ hardware through the D3D11 DDI. WDDM1.3 drivers on Feature Level 10+ hardware can optionally support this feature through the D3D9 DDI; however, such support should be consistent with its support for all other Optional Performance Features (e.g. Shared Surface Support).

To ensure the desired performance characteristics when Present1 is used, the following is required:

- Rendering as a result of calling the new DDI must be same as the rendering as a result of using traditional multiple single-present DDI calls.
- The driver must only call the Present callback once per DDI call (and not once per resource).

Device.Graphics.WDDM13.Render.SharedSurfaceSupport

Shared Surface Support

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

WDDM1.3 drivers must support these new formats and capabilities for shared surfaces:

- Drivers support new formats on Level 9 hardware,
  - A8_UNORM
  - R8_UNORM
  - R8G8_UNORM
- Driver supports shared surfaces in new formats (A8_UNORM, R8G8_UNORM, R8_UNORM, BC1_*, BC2_*, and BC3_*) with the following capabilities on Feature Level 9 hardware through the D3D9 DDI, as well as on Feature 10+ hardware through the D3D11 DDI.

WDDM1.3 drivers on Feature Level 10+ hardware can optionally support this feature through the D3D9 DDI; however, such support should be consistent with its support for all other Optional Performance Features (e.g. Present Overhead Support).

- Resources sharing;
- Resources can be used as render target;
- Resources can be blended;
- Resources can be filtered;
- Resources can be accessed.

Device.Graphics.WDDM13.DisplayRender.MultiplaneOverlaySupport

Multi-plane Overlay support

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All WDDM 1.3 display drivers must report multi-plane overlay capabilities of the hardware using the DDI defined by the Multi-plane Overlay Support DDI specification. WDDM 1.3 drivers that claim to support multi-plane overlays must meet all of the following hardware criteria:

- Hardware must support non-overlapping planes.
    - One plane can cover one portion of the screen while another plane can cover a different, mutually exclusive, portion of the screen.
    - If there is any portion of the screen not covered by a plane, the hardware must scan out black for that area. The hardware can assume that there is a virtual plane at the bottom-most Z order that is filled with black.
- Hardware must support overlapping planes. Blending between the planes using pre-multiplied alpha must be supported. The hardware must also be able to enable/disable alpha blending on a per-plane basis.
- When only one output is active, the active output must support multi-plane overlays. In the case of clone mode where multiple outputs are simultaneously active, the hardware should

not report that it supports multi-plane overlays unless all active outputs support multi-plane overlays.

- The DWM's swapchain (plane 0) must be able to interact with the other overlay planes.
- All planes must be able to be enabled and disabled, including plane 0 (the DWM's swapchain).
- All planes must support source and destination clipping, including plane 0 (the DWM's swapchain).
- At least one plane must support shrinking and stretching, independent from other planes that may be enabled.
- Planes that support scaling must support both bilinear filtering and better than bilinear filtering quality.
- At least one plane must support:
  - Both 601 and 709 YUV to RGB matrix conversion for YUV formats.
  - Both normal range YUV luminance (16 - 235) and full range YUV luminance (0 – 255).
- The following register latching scenarios must be handled:
  - All per-plane attributes (buffer address, clipping, scaling, etc.) must atomically post on the vertical retrace period. When updating a block of registers, they must all post atomically (i.e. if the VSYNC occurs after writing 10 of 20 registers pertaining to the overlay plane, none of them will post until the next VSYNC since they cannot all post on the current VSYNC).
  - Each plane can be updated independently from the other planes. For example, if the plane 0 registers have been updated prior to the VSYNC and we update the plane 1 registers when the VSYNC occurs, the plane 1 updates might wait until the next VSYNC, but the plane 0 updates should occur on time.
  - When multiple planes are updated during a single present call, the updates should occur atomically. For example, if a single present call is updating plane 0 and enabling plane 1, the plane 0 registers should not post on the VSYNC unless the plane 1 registers also post on the same VSYNC.
- Transformation, scaling, and blending should occur as follows (in the specified order):

- The source allocation is clipped according to the specified source rectangle. The source rectangle is guaranteed to be bounded within the size of the source allocation.
- Apply Horizontal image flip, then Vertical image flip if requested.
- Apply scaling according to the destination rectangle, apply clipping according to the clip rectangle, and apply the appropriate filtering when scaling.
- Blend with allocations at other layers. Blending should be performed from top to bottom (or until hit opaque layer) in Z-order. If alpha blending is requested, hardware must honor the per-pixel-alpha and color value is pre-multiplied by alpha. Pseudo code is indicated below, this does “source over destination” operation repeatedly from top to bottom, (((Layer[0] over Layer[1]) over Layer[2]) over … Layer[n]). Outside of the destination rectangle, each layer must be treated as transparent (0,0,0,0).

```
Color = Color[0]; // layer 0 is topmost.
Alpha = Color[0].Alpha;
for (i = 1; Alpha < 1 && i < LayersToBlend; i++)
{
    Color += ((1 - Alpha) * Color[i]);
    Alpha += ((1 - Alpha) * Color[i].Alpha);
}
Output Color;
```

  It is OK if hardware blends bottom to top as long as the output result is the same. In this case, the following blend algorithm should be used:

```
Color = Color[LayersToBlend-1];  // Bottom most layer
Alpha = Color[LayersToBlend-1].Alpha;
if (LayersToBlend > 1)
{
    for (i = LayersToBlend - 2; Alpha < 1 && i >= 0; i--)
    {
        Color = Color[i] + (1 – Color[i].Alpha) * Color;
        Alpha = Color[i].Alpha + (1 – Color[i].Alpha) * Alpha;
    }
}
Output Color;
```

- Black color must be displayed at the area where not covered by any of destination rectangles from any layers. Hardware can assume there is a conceptual virtual black bottom most layer that is the size of the screen.

Device.Graphics.WDDM13.Render.TrimSupport

Direct3D Trim

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The Trim API is new in the WDDM1.3 updates to the D3D9 UM DDI. Graphics devices must support this API and the driver must do the following:

- After a device calls Trim, the graphics driver memory usage must return to within 5% of its usage after initial device creation.
- The responsiveness of the graphics driver for rendering operations after calling Trim must be comparable to that of the driver after initial Direct3D device creation.
- Calling Trim must not change the state of the Direct3D device – all rendering operations should continue to be evaluated normally.

Send comments about this topic to Microsoft

Device.Graphics.WDDM20

In this topic:

Device.Graphics.WDDM20.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- Core.MemoryResidencyModel

  A WDDM 2.0 driver must meet all requirements with defined in the WDDM 2.0 Specification.

Drivers must support the new model for managing memory residency for WDDM 2.0 as defined in the Memory Residency model specification.

Drivers must explicitly create one or more paging queues for each D3D Device. Synchronization of operations of the paging queues must be done using Monitored Fences. Asynchronous Trim callbacks can occur which the UMD must handle by evicting resources until memory usage is below the specified limit.

A residency list of resources in memory must be maintained by the UMD for each D3D Device. UMD must use Trim/Flush/Wait to make progress in case an application uses more than its allocated memory.

For compatibility with D3D11 and earlier versions, the UMD must manage residency based on Bind and Unbind operations.

UMD must handle Offer and Reclaim requests differently in order to correctly maintain the Residency list.

Feature Name	Applicable DX H/W Class	Full Graphics Driver	Render Only Driver	Display Only driver
MakeResident and Evict	All	Mandatory	Mandatory	N/A
Lock2 support	All	Mandatory	Mandatory	N/A
Monitored Fence support	All	Mandatory	Mandatory	N/A
GPU Sync Points	All	Mandatory	Mandatory	N/A
Trim Callback support	All	Mandatory	Mandatory	N/A
Offer/Reclaim support	All	Mandatory	Mandatory	N/A
MakeResident and Evict	All	Mandatory	Mandatory	N/A

- Virtual Memory:

A WDDM 2.0 driver must meet all requirements defined in the WDDM 2.0 Specification.

Drivers must support the new model for managing memory residency for WDDM 2.0.

All engines that implement GPU virtual addressing must support capabilities defined in the GPU Virtual Memory DDI Specification including setting relevant capability bits.

References to all resources must be via their virtual addresses and the virtual addresses must remain constant for the lifetime of the resources.

It is mandatory to support single GPU address space per GPU logical adapter per process.

Engines that support virtual addressing must support GPUMMU model for virtual addressing and must not have any dependency on physical addressing. Support for the IOMMU model of virtual addressing is optional.

Drivers must support multi-level page tables and must support a mixture of 4KB and 64KB page tables.

Display and Capture surfaces for integrated GPUs must be addressable via the Aperture Segment.

Display surfaces in discrete GPUs must be allocated contiguously from a memory segment.

Feature Name	Applicable DX H/W Class	Full Graphics Driver	Render Only Driver	Display Only
Virtual Paging support	All	If VA is implemented	If VA is implemented	N/A
Multi-level Page Table support	All	If VA is implemented	If VA is implemented	N/A
Side-by-side 4K/64K Page Tables	All	If VA is implemented	If VA is implemented	N/A
Page-based Memory Segment Management	All	If VA is implemented	If VA is implemented	N/A
Tiled Resources	DX11.1+	Mandatory	Mandatory	N/A
Programmable CPU Host Aperture support	All	If applicable	If applicable	N/A

LDA support	All	If applicable	If applicable	

- Map Default

  Display drivers for D3D11/D3D12 must meet all requirements defined in the **D3D 11/D3D12 Specifications**.

  When such pages are accessible by the CPU, they must be able to be marked as write-back or write-combine for the CPU. Memory used by default, CPU-accessible textures must be marked as write-back if CPU-read is enabled. If CPU-write is enabled the memory must be marked as write-combine, except on cache coherent UMA designs.

  Pitch-linear requirements & restrictions are as follows:

  - The base address/ offset of mapped resources must be at least 16-byte aligned.
  - Compressed render targets must be able to be resolved to enable the CPU to read/ write the texels.

  Memory requirements & restrictions are as follows:

  - The base address/ offset of mapped resources must be at least 16-byte aligned.
  - Compressed render targets must be able to be resolved to enable the CPU to read/ write the texels.
  - The texture must be resident in system memory
  - The texture must not be renamed or shadow-copied as a result of mapping
  - Swizzle ranges must not be used while the texture is mapped

- When the GPU supports standard swizzle, the GPU must support all multi-dimensional operations as orthogonally as other textures, such as texture from, render to, copy to & from, re-swizzle to & from, etc.
  - Element size for block-compressed formats and ASTC formats must be same as the block size.
  - GPU must support standard swizzle from all engines, including scan-out.
  - Compressed render targets must be able to be resolved to enable the CPU to read/ write the texels.

- Standard swizzle data must be aligned to 4KB and 64KB page boundaries.

Business Justification:

WDDM 2.0 introduces new set of features and DDI changes designed to work optimally with next version of Windows.

Send comments about this topic to Microsoft

Device.Graphics.WDDM20.Core

The base feature set implemented by drivers supporting WDDM2.0.

In this topic:

Device.Graphics.WDDM20.Core.Base [If Implemented]

WDDM2.0

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Graphics drivers can optionally implement WDDM 2.0 if they do not support D3D12. However, D3D12 support requires WDDM 2.0.

Windows 10 introduces a new set of features and DDI changes that will be referred to as WDDM2.0. The implementation of WDDM2.0 is optional although it is required for implementing a driver that supports D3D12. If implemented, the WDDM2.0 specification must be followed.

D3D Requirements:

DirectX Hardware	KMD Requirements	UMD Requirements
D3D9	Required: WDDM 2.0	Required: D3D9 - UMD DDI
D3D10	Required: WDDM 2.0	Required: D3D9 - UMD DDI
		Required: D3D10 - UMD DDI

		Required: D3D11.1 - UMD DDI
D3D10.1	Required: WDDM 2.0	Required: D3D9 - UMD DDI
		Required: D3D10 - UMD DDI
		Required: D3D10.1 - UMD DDI
		Required: D3D11.1 - UMD DDI
D3D11	Required: WDDM 2.0	Required: D3D9 - UMD DDI
		Required: D3D10 - UMD DDI
		Required: D3D10.1 - UMD DDI
		Required: D3D11 - UMD DDI
		Required: D3D11.1 - UMD DDI
D3D11.1	Required: WDDM 2.0	Required: D3D9 - UMD DDI
		Required: D3D10 - UMD DDI
		Required: D3D10.1 - UMD DDI
		Required: D3D11 - UMD DDI
		Required: D3D11.1 - UMD DDI
D3D12	Required: WDDM 2.0	Required: D3D9 - UMD DDI
		Required: D3D10 - UMD DDI
		Required: D3D10.1 - UMD DDI
		Required: D3D11 - UMD DDI
		Required: D3D11.1 - UMD DDI

		Required: D3D12 - UMD DDI

Send comments about this topic to Microsoft

Device.Graphics.WDDM20.Display.VirtualModeSupport

In this topic:

Device.Graphics.WDDM20.Display.VirtualModeSupport.CoreRequirement [if implemented]

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

WDDM2.0 drivers must report support for virtual resolution changes and DWMClone, via the new VirtualModeSupport member in DXGK_DISPLAY_DRIVERCAPS_EXTENSION, by setting VirtualModeSupport to 1 when the OS calls DxgkDdiQueryAdapterInfo on DXGKQAITYPE_DISPLAY_DRIVERCAPS_EXTENSION.

Send comments about this topic to Microsoft

Device.Graphics.WDDM20.DisplayRender

The parent feature node of requirements that have an impact for the display and render GPU.

In this topic:

Device.Graphics.WDDM20.DisplayRender.HardwareContentProtection

A GPU must support hardware-based content protection.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The GPU driver for the next version of Windows must adhere to the Graphics – D3D11 Content Protection DDI Specifications.

The GPU driver and hardware must pass the Microsoft Hardware Content Protection tests.

Requires a network connection and access to PlayReady Content Protection test servers.

Send comments about this topic to Microsoft

Device.Imaging.Printer.Base

In this topic:

Device.Imaging.Printer.Base.applicationVerifier

Printer driver components must comply with Application Verifier test criteria.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

All user-mode modules (.dll or .exe files) that are part of a printer driver must satisfy the criteria for Application Verifier tests. During driver testing, Application Verifier must be turned on for processes in which the driver modules execute. Application Verifier causes a break when Application Verifier detects an error.

For printer drivers, common applications that must have Application Verifier turned on during testing are the following:

- Splwow64.exe.
- Spoolsv.exe. The process loads the rendering and UI portions of the driver during print testing.
- Printfilterpipelinesvc.exe. The process loads the XPS rendering filters.
- Any vendor-supplied applications that are part of the driver package, such as a custom status monitor. All portions of the driver package that is being signed must be robust.
- Any tests that load driver modules for rendering or UI purposes. The tests commonly load UI and rendering modules.

The following Application Verifier tests must be turned on for these processes during testing:

- Heap
- Locks
- Handles
- FilePaths
- HighVersionLie
- DFWChecksNonSetup
- SecurityChecks
- Printing

Design Notes:

For information about Application Verifier, see http://go.microsoft.com/fwlink/?LinkId=58371.

Device.Imaging.Printer.Base.configurationFiles

Version 4 print drivers must provide valid configuration files.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Version 4 print drivers must provide valid configuration files.

- These files must be syntactically valid according to the WDK.
- When supported by the hardware, the configuration files must support the following features:
  - Orientation
  - Duplexing
  - Collate
  - N-Up
  - Paper Size
  - Paper Type
  - Tray
  - Quality
  - Color
  - Stapling
  - Hole-punches
  - Binding
- GPD or PPD files should define the majority of the features and constraints represented in the driver's PrintCapabilities. JavaScript implementations should supplement these capabilities as needed.
- JavaScript files must be syntactically valid according to the WDK.
  - They must be included in the driver package and cannot be in a subdirectory in the package.

- They may be included only with version 4 print drivers.
- They should be designed securely and validate any untrusted data that will be parsed; this includes PrintCapabilities, PrintTicket, XPS Documents, and Property Bags.

Device.Imaging.Printer.Base.connectionRecovery

A printer must continue to operate normally if a computer becomes unavailable during a print job.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a computer becomes unavailable during a print job (for example, if the computer enters a sleep state or a network failure or other event interrupts the connection between the computer and printer), the printer must recover so that normal print operations can continue without user interaction.

Design Notes:

Specifically, the printer must not fail into a state in which the end user must manually power cycle the printer or clear a paper jam.

The printer does not have to complete or continue the failed print job when the computer becomes available again. However, when computer-to-printer communication is restored, new print jobs must be able to spool and complete normally.

Device.Imaging.Printer.Base.deviceCapabilities

A printer must correctly support the DeviceCapabilities and GetDeviceCaps application programming interfaces (APIs) based on the guidelines in the Windows Driver Kit.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement clarifies the use of existing HLK tests. The Print Driver Device Capabilities test determines whether the printer driver returns the correct information for the **DeviceCapabilities** and **GetDeviceCaps** API calls.

For more information, see http://msdn.microsoft.com/en-us/library/dd183552(v=vs.85).aspx

and http://msdn.microsoft.com/en-us/library/ff566075(v=VS.85).aspx.

Device.Imaging.Printer.Base.driverCategory

Version 4 printer drivers must declare a valid printer category.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All V4 printer drivers must declare a valid DriverCategory in their driver manifest.

V3 printer drivers are not required to declare a category. If a V3 printer driver does declare a DriverCategory, it must be valid value.

The DriverCategory must be one of the following values:

- PrintFax.Printer
- PrintFax.Fax
- PrintFax.Printer.File
- PrintFax.Printer.Virtual
- PrintFax.Printer.Service

Device.Imaging.Printer.Base.DriverPDC [If implemented]

INF files must have correct syntax when PWG Raster rendering filter is included

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description:

All v4 print drivers which make use of the PrintDeviceCapabilities DataFile type must implement their PrintDeviceCapabilities file properly. Drivers which make use of the PWG Raster standard rendering filter must provide a PrintDeviceCapabilities DataFile which conforms to this requirement.

Device.Imaging.Printer.Base.GDLFile

GDL files must use the correct syntax according to the guidelines in the Windows Driver Kit.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

This requirement clarifies the use of existing HLK tests. The Generic Description Language (GDL) Correctness Test determines whether GDL files use the correct syntax according to the WDK.

For more information, see http://msdn.microsoft.com/en-us/library/ff549787(v=VS.85).aspx and http://msdn.microsoft.com/en-us/library/ff563355(v=VS.85).aspx.

Device.Imaging.Printer.Base.infFile

INF files must use the correct syntax according to the guidelines in the Windows Driver Kit.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement clarifies the use of existing HLK tests.

INFGate determines whether INF files and v4 manifest use the correct syntax according to the WDK.

Version 4 print drivers use version 4 INFs and manifest files.

V4 driver INF must:

- Define a PrinterDriverID for each driver in the INF to indicate when drivers are compatible for the sake of printer sharing.
- PrinterDriverID must be a GUID.
- Set a DataFile as a GPD or PPD file.
- Use only the following DestinationDirs: DriverStore, 66000; or Color directory, 66003.
- Specify a filter xml pipeline config file named *-pipelineconfig.xml, OR specify RequiredClass in the v4 driver manifest.

V4 driver manifest files must:

- Define a PrinterDriverID for each driver in the manifest to indicate when drivers are compatible for the sake of printer sharing.
- PrinterDriverID must be a GUID.
- Set a DataFile as a GPD or PPD file.

V4 drivers must not:

- Utilize any of the following INF directives ClassInstall32, ClassInstall32.Services, DDInstall.Services, DDInstall.HW, DDInstall.CoInstallers, DDInstall.FactDef, DDInstall.LogConfigOverride, DDInstall.Interfaces, InterfaceInstall32, DefaultInstall, DefaultInstall.Services, AddReg, DelReg, DelFiles, RenFiles, AddService, DelService, AddInterface, BitReg, LogConfig, UpdateInis, UpdateIniFields, or Ini2Reg.

Version 3 INF files must use correct syntax according to the WDK.

For more information on v3 INF files, see http://msdn.microsoft.com/en-us/library/ff560902(v=VS.85).aspx and http://msdn.microsoft.com/en-us/library/ff563414(v=VS.85).aspx.

Device.Imaging.Printer.Base.printProcessor

Print processors must be implemented based on the guidelines in the Windows Driver Kit.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement clarifies the use of existing HLK tests. The Print Processor API test helps to ensure that print processors are implemented based on WDK guidelines.

All print processors must support the following endpoints:

- OpenPrintProcessor
- ClosePrintProcessor
- ControlPrintProcessor
- EnumPrintProcessorDatatypesW
- PrintDocumentOnPrintProcessor
- GetPrintProcessorCapabilities

For more information, see http://msdn.microsoft.com/en-us/library/ff566084(v=VS.85).aspx.

Device.Imaging.Printer.Base.printRegions

A printer must support printable regions accurately.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The printer must be able to print output for the entire area that appears in the printable region that the user can select in the Windows UI.

Design Notes:

Note that if the printer supports a "borderless" printing feature, this restriction may be relaxed to allow for devices whose printable region extends beyond the dimensions of the physical media sheet. In these cases, the printer must be able to print output to the extent of the page dimension.

This test applies to all paper sizes that the printer physically supports. If the printer supports auto-scaling and the UI exposes additional paper sizes to the user that cannot be physically loaded into the printer, the printer must maintain correct aspect ratios during resizing. In this context, "auto-scaling" is any feature in the hardware or driver that changes the print job to fit on an available paper size without user interaction or warning.

If the printer does not support printing on a physical medium that is at least 4" x 4" in size, the printer is exempt from this requirement.

Device.Imaging.Printer.Base.printTicket

A printer driver must support PrintTicket/PrintCapabilities.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Print devices must fully support the **PrintTicket** and **PrintCapabilities** objects. Depending on your print driver implementation, this may or may not require implementing certain **PrintTicket** or **PrintCapabilities** interfaces. For more information, see the WDK documentation.

Printer drivers must support the public Print Schema element types for each keyword if the printer driver or print device includes the specified functionality. Printer drivers must also support the public Print Schema element types for each keyword if the appropriate functionality is present but non-configurable. The element types that the printer driver must support for each keyword include all such Features, Options, ParameterDef, ParameterRef, Property, and ScoredProperty that the public Print Schema definition contains. Printer drivers do not have to support public Print Schema keywords if the printer driver or print device does not include the specified functionality.

Printer drivers must support the following basic Print Schema element types:

- DocumentCollate
- JobCopiesAllDocuments

- JobDuplexAllDocumentsContiguously
- PageColorManagement
- PageImageableSize
- PageMediaSize
- PageMediaType
- PageOrientation
- PageOutputColor
- PageResolution
- PageICMRenderingIntent
- One of the following: JobInputBin, DocumentInputBin, or PageInputBin

Device.Imaging.Printer.Base.rendering

A printer must correctly render print jobs.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement clarifies the use of the following existing HLK tests to ensure that printers correctly render print jobs:

- Pgremlin/Pgremlin2

This test produces numerous pages of output that include shapes, gradient fills, alpha blends and some complex fonts. The test checks Device Driver Interface (DDI) calls that a driver can make.

- WinFX Markup Content Rendering Test

The WinFX Markup Content Rendering test loads eight WinFX XAML files and produces output on the specified printer.

- Photo Print Test

The Photo Print Test prints landscape- and portrait-oriented photos to exercise printer functionality.

For more information about Pgremlin, see http://msdn.microsoft.com/en-us/library/ff566081(v=VS.85).aspx.

For more information about Pgremlin2, see http://msdn.microsoft.com/en-us/library/ff566079(v=VS.85).aspx.

For more information about the WinFX Markup Content Rendering Test, see http://msdn.microsoft.com/en-us/library/ff569275(v=VS.85).aspx.

For more information about the Photo Print Test, see http://msdn.microsoft.com/en-us/library/ff565234(v=VS.85).aspx.

Device.Imaging.Printer.Base.TCPMon

Network-connected printers that support Port 9100 printing with the Microsoft Standard Port Monitor (TCPMon) must provide rich status for the device.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the printer uses a network interface port connection and supports Port 9100 printing (raw printing) with the Microsoft Standard Port Monitor (TCPmon), it must also support Simple Network Management Protocol (SNMP), Host Resource Management Information Base (MIB), and Common Printer MIB, and Printer Port Monitor MIB 1.0 (IEEE-ISTO5107.1-2005) so that the operating system can provide rich status for the device.

Send comments about this topic to Microsoft

Device.Imaging.Printer.Mobile

In this topic:

Device.Imaging.Printer.Mobile.Mobile [If implemented]

Printer must implement the following items in the description to ensure proper functionality

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64
	Windows Server 2016 Technical Preview

Description:

- Printers must implement the "MobilePrint" category definition in their WS-Discovery message's PnP-X DeviceCategory element.

Devices which support Windows Mobile Printing must add the "MobilePrinter" category to their WS-Discovery ThisModel response.

The following table provides additional information about the MobilePrinter category keyword.

Constant/Value	Description
PNPX_DEVICECATEGORY_PRINTER_MOBILE L"MobilePrinter"	MobilePrinter category Keywords: Printer

- V4 print drivers that use PrintDeviceCapabilities must ensure that it is correctly defined by including a PrintDeviceCapabilities DataFile.
- If a v4 print driver includes the PWG Raster standard rendering filter, then a PrintDeviceCapabilities DataFile must be included.

Device.Imaging.Printer.Mobile.PDL [If implemented]

Printers with mobile print capability must support one of the following PDLs and conform to the relevant requirements for it.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

PWG-Raster

Printers supporting PWG Raster must conform to the PWG Raster industry standard and implement the following requirements.

- Follow all the rules described in PWG 5102.4-2012.
- Devices supporting duplex must support InsertPage directive.
- Devices must consume all color formats and resolutions that are advertised in PrintDeviceCapabilities file.

- Devices must support all advertised psk:PageMediaSizes from the PrintDeviceCapabilities file when encoded in to the PWG Raster Page Header according to PWG 5101.1's definition of self-describing media names.
- Devices must support one of the following color formats
  - Srgb_8
  - Cmyk_8
  - Sgray_8
- Devices supporting WS-Print v2.0 must indicate support for at least one psk:PageResolution which is less than or equal to 360dpi.
- For mobile compatible mode, device must support 300 DPI output
- For mobile compatible mode, device must accept Letter or A4 formats.

PCLm

Printers supporting PCLm must conform to industry standard and implement the following requirements.

- Device must support the default settings defined in section 4.2.9 of the Wi-Fi Direct Services Print Technical Specification v1.0:

Attribute	**Default setting**
StripHeight	16
Media	Letter
Resolution	600
Duplex	Simplex
ColorSpace	sRGB
Copies	1
Finishings	None
Fit-to-page	False

Orientation	Portrait
PrintQuality	Normal

- Device must support receiving content at 300 dpi
- Device must support receiving content compressed using flate, run-length encoding or DCTDecode

OXPS

Printer supporting OpenXPS must conform to industry standard and implement the following requirements.

- If the device implements OpenXPS support, it must follow all rules described in ECMA-388
- Device must process PrintTickets in the following order of precedence (most authoritative -> least authoritative):
  - Page-level PrintTicket embedded in document
  - Document-level PrintTicket embedded in document
  - CreatePrintJob2 Job-level PrintTicket
  - Job-level PrintTicket embedded in document

Microsoft XPS

Printer supporting Microsoft XPS must conform to documented specification.

- Printer supporting Microsoft XPS must conform and follow all rules described in the Microsoft XML Paper Specification 1.0

Send comments about this topic to Microsoft

Device.Imaging.Printer.Mobile.WSD20

In this topic:

Device.Imaging.Printer.Mobile.WSD20.PDC [If implemented]

Printer supporting PrintDeviceCapabilities (PDC) must support the following items in the description

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description:

- Printers supporting PrintDeviceCapabilities must conform with all of the rules set out in the Print Schema Specification v2.0, as well as the WDK.
- Only currently available features may be exposed in a PrintDeviceCapabilities file. Printers must describe their currently available Features, Options, and Parameters in a PrintDeviceCapabilities file.

  If any Features, Options or Parameters are conditionally available, depending on the state of an installable option such as a finisher, then these items must only be exposed in the PrintDeviceCapabilities file if the pre-requisite installable option is currently installed.
- All options in PageMediaSize must be exposed in PrintDeviceCapabilities files. Printers must describe all supported paper sizes using the psk:PageMediaSize feature in a PrintDeviceCapabilities file.
- All options in PageResolution must be exposed in PrintDeviceCapabilities files. Printers must describe their supported device resolutions using the psk:PageResolution feature in a PrintDeviceCapabilities file.
- Printers must describe at least one input bin using either the psk:JobInputBin, psk:DocumentInputBin, or psk:PageInputBin features.
- Printers supporting PWG Raster must contain PwgRasterDocumentTypesSupported element in the PrintDeviceCapabilities file to describe the supported color formats.
- Printers which indicate the deviceProcessing attribute in a PrintDeviceCapabilities file must support those features without interaction of the host. If the printer indicates that an Option or Parameter is processed in hardware using the deviceProcessing="true" attribute, then it MUST handle that Option or Parameter without the participation of the host.

Device.Imaging.Printer.Mobile.WSD20.WSD20support [If implemented]

Printer supporting WS-Print 2.0 must implement the following requirements

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

- Printers supporting WS-Print v2.0 must describe and support the following MIME types, by including the wprt2:PrinterFormats element under the printer's wprt:PrintCapabilities element.
  - PrintDeviceCapabilities
    - application/vnd.ms-PrintDeviceCapabilities+xml
  - PrintJobTicket
    - application/vnd.ms-PrintSchemaTicket+xml
  - PrintDeviceResource
    - application/vnd.ms-resx+xml
- Printers supporting WS-Print v2.0 must handle PrintDeviceCapabilitiesChangeID. Printers must describe the PrintDeviceCapabilitiesChangeID element in the wprt:PrinterConfiguration element and create an event if the current PrintDeviceCapabilitiesChangeID changes.
  - The PrintDeviceCapabilitiesChangeID must be consistent across power cycles of the device unless the state of the printer changes (eg, an installable option is added or removed).
- Printer supporting WS-Print v2.0 must support one of the following PDLs by describing the corresponding MIME type in the wprt:PrinterCapabilities wprt:Format element.
  - application/oxps
  - application/vnd.ms-xpsdocument
  - image/pwg-raster
  - application/pclm

- Printers supporting WS-Print v2.0 must support retrieval of resource files in the following format using the GetPrintDeviceResources operation, or must return the NoLocalizedResources SOAP fault.
  - application/vnd.ms-resx+xml
- Printer's WS-Print v2.0 implementation must support all new Operations described in the WS-Print v2.0 specification:
  - CreatePrintJob2
  - PrepareToPrint
  - GetBidiSchemaExtensions
  - GetPrintDeviceResources
  - GetPrintDeviceCapabilities
- Printers supporting WS-Print v2.0 must process the PrintJobTicket element if it is included in the CreatePrintJob2 request. Printers must support PrintJobTickets formatted in the following format.
  - application/vnd.ms-PrintSchemaTicket+xml
- Printers supporting Windows Mobile must implement WS-Print v2.0 in accordance to guidance in WDK.

Send comments about this topic to Microsoft

Device.Imaging.Printer.OXPS

In this topic:

Device.Imaging.Printer.OXPS.OXPS

V4 drivers that support OpenXPS must be implemented according to the rules specified in the WDK.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

For Windows 10, a correctly implemented version 4 print driver will satisfy the XPS requirements. The v4 driver can support either MSXPS or Open XPS.

V4 print drivers that support OpenXPS, either exclusively or in dual-format support mode with XPS, must satisfy the following requirements:

- Driver manifest must specify either "XpsFormat=OpenXPS", "XpsFormat=OpenXPS, XPS" or "XpsFormat=XPS, OpenXPS" in the DriverRender section.
- The first filter must be able to consume OpenXPS document format when provided such by the print filter pipeline manager.
- All filters must conform to the rendering rules in the ECMA Open XML Paper Specification v. 1.0 (Ecma 388).
- All filters must conform to the PrintTicket processing rules in the PrintSchema Specification 1.0.
- The v4 driver filter pipeline must produce a PDL that the target print device can interpret.
- Filters in the v4 driver pipeline that support OpenXPS must NOT do the following:
  - Call into the Common Language Runtime (CLR) or the WinFX run-time components for any functionality.
  - Display user interface (UI) content.
- OpenXPS supporting drivers must adhere to all other v4 rules. Dual-format drivers must adhere to both OpenXPS and MSXPS requirements and successfully handle either format.

Send comments about this topic to Microsoft

Device.Imaging.Printer.USB

In this topic:

Device.Imaging.Printer.USB.CompatID

Printers must implement a Compatible ID in their IEEE1284 string according to the rules specified in the WDK.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Printers must implement a Compatible ID in their IEEE1284 string for devices that connect over USB and WSD using the Port Monitor MIB.

The Compatible ID must indicate:

- Manufacturer Name or Code
- Device Class Description
    - Recommended:
        - Include PDL
        - any other relevant device class information
        - Example: Fabrikam_XPS_A3_laser

Devices that receive the Windows 7 logo before June 1, 2012 are exempt from this requirement.

Link to Compatible ID Whitepaper: http://msdn.microsoft.com/en-us/windows/hardware/gg463313.aspx

Device.Imaging.Printer.USB.JSBidiExtender

USB JavaScript Bidi Extenders are implemented according to the guidance in the WDK.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB JavaScript Bidi Extenders are implemented according to the guidance in the WDK:

- They must be included in the driver package and cannot be in a subdirectory in the package.
- They may only be included with version 4 print drivers.

- They should be designed securely and validate any untrusted data that will be parsed.
- They must be referenced in the v4 manifest files.

They must use syntactically valid JavaScript, implemented according to the WDK.

Send comments about this topic to Microsoft

Device.Imaging.Printer.WSD

In this topic:

Device.Imaging.Printer.WSD.CompatID

Printers must implement a Compatible ID in their IEEE1284 string according to the rules specified in the WDK.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Printers must implement a Compatible ID in their IEEE1284 string for devices that connect over USB and WSD using the Port Monitor MIB.

The Compatible ID must indicate:

- Manufacturer Name or Code
- Device Class Description
    - Recommended:
        - Include PDL
        - any other relevant device class information
        - Example: Fabrikam_XPS_A3_laser

Link to Compatible ID Whitepaper: http://msdn.microsoft.com/en-us/windows/hardware/gg463313.aspx

Send comments about this topic to Microsoft

Device.Imaging.Printer.XPS

In this topic:

Device.Imaging.Printer.XPS.XPS

A printer must have a driver that correctly implements the XPSDrv printer driver architecture.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The v4 driver can support either MSXPS or Open XPS.

V4 print drivers that support MSXPS, either exclusively or in dual-format support mode with Open XPS, must satisfy the following requirements:

- Driver manifest must specify either "XpsFormat=OpenXPS", "XpsFormat=OpenXPS, XPS" or "XpsFormat=XPS, OpenXPS" in the DriverRender section.
- The first filter must be able to consume the XPS document format when provided such by the print filter pipeline manager.
- All filters must conform to the rendering rules in the XML Paper Specification.
- All filters must conform to the PrintTicket processing rules in the PrintSchema Specification 1.0.
- The v4 driver filter pipeline must produce a PDL that the target print device can interpret.
- Filters in the v4 driver pipeline supporting XPS must NOT do the following:
  - Call into the Common Language Runtime (CLR) or the WinFX run-time components for any functionality.
  - Display user interface (UI) content.
- XPS supporting drivers must adhere to all other v4 rules. Dual-format drivers must adhere to both OpenXPS and MSXPS requirements and successfully handle either format.

A printer must have a driver that correctly implements the XPSDrv printer driver architecture.

A driver that implements the XPSDrv printer driver architecture must not do the following:

- Implement a GDI rendering module.
- Implement a print processor.

If the XPSDrv driver supports a print device that can consume the XPS Document format as a printer description language (PDL), no filters are required. Otherwise, or if the driver supplies filters, the driver must satisfy the following requirements:

- The first filter in the XPSDrv driver filter pipeline must consume the XPS Document format.
- The XPSDrv driver filter pipeline must produce a PDL that the target print device can interpret.
- Filters in the XPSDrv driver filter pipeline must do the following:
  - Conform to the rendering rules in the XML Paper Specification.
  - Conform to the PrintTicket processing rules in the XML Paper Specification.
- Filters in the XPSDrv driver filter pipeline must not do the following:
  - Call into the Common Language Runtime (CLR) or the WinFX run-time components for any functionality.
  - Display user interface (UI) content.

XPSDrv must fully support the PrintTicket and PrintCapabilities objects. XPSDrv drivers must also support the following additional keywords in the described under the printTicket requirement.

- PageScaling
- JobDigitalSignatureProcessing
- PagePhotoPrintingIntent
- PageOutputQuality

Send comments about this topic to Microsoft

Device.Imaging.Scanner.Base

In this topic:

- Device.Imaging.Scanner.Base.wia20
- Device.Imaging.Scanner.Base.WIAProperties

Device.Imaging.Scanner.Base.dataTransfer

WIA drivers must support specific data transfer implementations.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows Image Acquisition (WIA) drivers must do the following:

- Use only the Write, Seek, and SetSizeIStream methods during downloads.
- Use only the Read, Seek, and StatIStream methods during uploads.
- Send valid lPercentComplete values during calls to the IWiaMiniDrvTransferCallback::SendMessage<element>. lPercentComplete must be between 0 and 100 inclusive.
- Send correct ulTransferredBytes values during calls to IWiaMiniDrvTransferCallback::SendMessage.
- Append new data to the end of streams that the IWiaMiniDrvTransferCallback::GetNextStream<element> receives if the streams are not empty.
- Return success values from the IWia MiniDrv::drvAcquireItemData method when the driver calls IWiaMiniDrv::drvAcquireItemData by using good parameters in all supported formats.
- Release their references to the application's IStream object before their IWiaMiniDrv::drvAcquireItemData methods return or call IWiaMiniDrvTransferCallback::GetNextStream.
- Send one stream that contains a multi-page file during downloads by using all supported TYMED_MULTIPAGE_FILE formats.
- Send one stream for each item during downloads by using all supported TYMED_FILE formats.

- Return S_FALSE from IWiaMiniDrv::drvAcquireItemData if IWiaMiniDrvTransferCallback::SendMessage returns S_FALSE.
- Continue to transfer any subsequent items during multi-item downloads after IWiaMiniDrvTransferCallback::GetNextStream returns WIA_STATUS_SKIP_ITEM.
- Return S_OK from IWiaMiniDrv::drvAcquireItemData during single-item and multi-item downloads after the IWiaMiniDrvTransferCallback::GetNextStream returns WIA_STATUS_SKIP_ITEM for any item.
- Abort transfer of all items after IWiaMiniDrvTransferCallback::GetNextStream returns S_FALSE.
- Return from IWiaMiniDrv::drvAcquireItemData calls during canceled transfers in less time than during completed transfers.
- Return a failure code from IWiaMiniDrv::drvAcquireItemData if IWiaMiniDrvTransferCallback::GetNextStream fails.
- Return a standard COM failure code from IWiaMiniDrv::drvAcquireItemData if IWiaMiniDrvTransferCallback::GetNextStream returns a NULL stream pointer.
- Return a failure code from IWiaMiniDrv::drvAcquireItemData if IWiaMiniDrvTransferCallback::SendMessage fails.
- Successfully transfer items when an identical device is installed and when the identical device transfers an item simultaneously.
- Return a failure code from IWiaMiniDrv::drvAcquireItemData if an IStream method fails.
- Seek to the correct stream position after the driver begins the transfer or calls IWiaMiniDrvTransferCallback::GetNextStream or IWiaMiniDrvTransferCallback::SendMessage.
- Function correctly if the WIA service terminates during a transfer and is then restarted, and a new transfer is requested.
- Ignore calls to the drvNotifyPnPEvent<element> that contain WIA_EVENT_CANCEL_IO if a transfer is not occurring, and not crash or hang.
- Successfully transfer items after a valid WIA_IPA_TYMED property value is written by using a signed or unsigned variant type.

WIA drivers must not do the following:

- Call a method of an application's IStream object other than the IUnknown::Release method after an application's transfer callback returns S_FALSE, WIA_STATUS_SKIP_ITEM, or an error.
- Call a method of an application's IStream object other than the IUnknown::Release method after the driver's IWiaMiniDrv::drvAcquireItemData method returns or calls IWiaMiniDrvTransferCallback::GetNextStream during a multi-item transfer.
- Call IWiaMiniDrvTransferCallback::SendMessage by using WIA_TRANSFER_MSG_END_OF_STREAM and WIA_TRANSFER_MSG_END_OF_TRANSFER messages.
- Call IWiaMiniDrvTransferCallback::SendMessage or IWiaMiniDrvTransferCallback::GetNextStream after IWiaMiniDrvTransferCallback::SendMessage returns S_FALSE or an error.
- Crash or hang if a device requests an upload by using an invalid or empty stream.

Device.Imaging.Scanner.Base.wia20

Scanners and multi-function printers that have scanning ability must implement the WIA 2.0 driver architecture according to the Windows Driver Kit guidelines.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Scanners and multi-function printers that have scanning ability must implement the WIA 2.0 driver architecture according to the guidelines in the Windows Driver Kit.

Scanners support stream-based image transfers. In stream-based transfers, the application gives WIA the stream to use, and then the driver reads or writes to the stream. The stream may be a file stream, a memory stream, or any other type of stream. The stream is transparent to the driver. Using streams also provides easy integration with the image processing filter. This helps to prevent complications that occur because of the destination type (memory or file).

Scanners need to correctly implement the Windows WIA Scanner Item Architecture for flatbed, ADF, and film scanners and scanners that have storage. The Windows Driver Kit reference documents and tools outline the WIA scanner item architecture. Device manufacturers must implement WIA support as described in the Windows Driver Kit.

Design Notes:

WIA 2.0 enables new stream-based transfer models and certain extensions that include an image-processing filter, a segmentation filter, and error handling. For more information about WIA 2.0, see "Introduction to WIA 2.0" at http://www.microsoft.com/whdc/device/stillimage/WIA20-intro.mspx and "What's new in WIA 2.0" at http://msdn.microsoft.com/en-us/library/ms630379(VS.85).aspx.

Device.Imaging.Scanner.Base.WIAProperties

Scanners must implement support for all required WIA properties and property values.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The Windows Driver Kit (WDK) reference documents and tools outline the properties and property values for WIA. Scanners must implement WIA as described in the WDK.

Send comments about this topic to Microsoft

Device.Imaging.Scanner.WSD

In this topic:

Device.Imaging.Scanner.WSD.WSScan

Scanners that have a network connection must implement the WS-Scan protocol.

Applies to	Windows 10 x64 Windows 10 x86

Description

This requirement applies to scanners or multifunction printers that have scanning ability, connect to the network, and have a physical scan button on the device. These scanners or multifunction printers must implement the following WS-Scan protocol requirements as outlined in the WS-Scan specification v1.0:

- The device must correctly implement all events and operations that the specification defines.

- The device must support the ScanAvailableEvent so that users can initiate a scan from the scanner and push the document to a scanning application.

- The device must support the Microsoft WSD scan class driver for all device features that WS-Scan exposes.

- If the device has ADF and plate scanning, the device must support those media over WS-Scan.

For more information, see "Implementing Web Services on Devices for Printing and Scanning" at http://www.microsoft.com/whdc/connect/rally/wsdspecs.mspx.

Send comments about this topic to Microsoft

Device.Input.Digitizer.Base

In this topic:

Device.Input.Digitizer.Base.ContactReports

Digitizer reliability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.NoiseSupression**

A digitizer shall only report data corresponding to user input. No other data, often referred to as "Phantom" or "Ghost contacts" shall be reported. This applies both when the system is actively receiving user input and when it is not receiving user input and under both AC and DC power conditions (where applicable).

Device.Input.Digitizer.Base.HIDCompliant

HID compliant device firmware and/or HID mini-port driver

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.HIDCompliantFirmware**

All digitizers are required to be human interface device (HID) compliant and adhere to the specification for their respective usage in order to report input to Windows.

Please see the Device.Input.Digitizer.Base.ThirdPartyDrivers for additional information on use of inbox HID mini-port drivers vs 3rd party HID mini-port drivers.

For more information on implementation, see:

For the reporting requirements, see the collection of topics at http://msdn.microsoft.com/en-us/library/windows/hardware/jj151569(v=vs.85).aspx

For HID over I2C requirements, see http://msdn.microsoft.com/en-us/library/windows/hardware/Dn642101.aspx

For HID over USB requirements, see http://www.usb.org/developers/devclass_docs/HID1_11.pdf

For HID over Bluetooth requirements, see http://developer.bluetooth.org/TechnologyOverview/Documents/HID_SPEC.pdf

Business Justification

The Human Interface Device (HID) protocol is the industry recognized standard that Microsoft requires for input devices. This requirement guarantees Windows compatibility and serviceability with any compliant digitizer solution.

Device.Input.Digitizer.Base.ThirdPartyDrivers

Servicing and 3rd party driver availability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Applicability: "Windows Desktop"

If a Windows Precision Touchpad digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM shipping image (and restore image) and available on Windows Update. Any 3rd party HID mini-port driver shall also be functional in WinPE. It is highly recommended that an inbox HID mini-port driver is utilized wherever possible for Windows Precision Touchpad implementations.

Special note: A touchpad may not be marketed as a Windows Precision Touchpad unless all the applicable compatibility requirements are met.

All Integrated Windows touch and pen devices shall be fully functional without the use of any 3rd party HID mini-port or filter drivers provided one is available for the associated bus used for connectivity. While Windows recommends that a device be shipped as tested, an OEM may elect to ship a Windows 10 compatible HID mini-port driver and/or filter driver that provides additional functionality after the device has met all associated HLK requirements including **Device.Input.Digitizer.Base.ThirdPartyDrivers** provided that all other requirements are met when that driver is installed. Any 3rd party HID mini-port driver which is used in instances where an inbox driver is not available for the bus selected for connectivity, that driver shall be part of the OEM shipping image (and restore image) and available on Windows Update. Any 3rd party HID mini-port driver shall also be functional in WinPE.

Applicability: “Windows Mobile”

If a Windows Precision Touchpad, integrated touch or integrated pen digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM Base Support Package (BSP) for the system and available on WU. It is highly recommended that an inbox HID mini-port driver is utilized wherever possible.

Business Justification

This requirement ensures that digitizers are always available for user input regardless of servicing scenarios and or clean installations. There are instances where the digitizer may be the only source of input for a given device

Send comments about this topic to Microsoft

Device.Input.Digitizer.Pen

In this topic:

- Device.Input.Digitizer.Pen.Resolution
- Device.Input.Digitizer.Pen.ThirdPartyDrivers

Device.Input.Digitizer.Pen.Accuracy

Pen contact accuracy

Target Feature	Device.Input.Digitizer.Pen

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The requirement Device.Digitizer.Pen.HoverAccuracy has been merged with this requirement.

When the pen tip is in contact with the screen, the delta between actual contact location and what the device reports shall be:

- <= +/- 0.5 mm (center)
- <= +/- 1 mm (within 3.5mm of all edges)

When hovering (with tip not in contact), the physical delta between physical location and what the device reports shall be <= +/- 1 mm (center).

Business Justification

This requirement ensures compatibility with DirectInk and inbox pen experiences; specifically:

- When the pen is in contact with the screen ensuring that ink appears to originate from the pen tip, and ensuring ability to target small controls.
- When the pen is hovering over the screen, ensuring that cursor position is perceived to be accurate by the user

Device.Input.Digitizer.Pen.Buffering

Buffering for buses with High Resume latency

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is an “If-Implemented” requirement. Active pen solutions that use a bus with high resume latency such as USB with selective suspend or Bluetooth shall implement an input report buffer capable of handling >= 100ms of data.

Input report buffer size (in bytes) = Input Report Size * (100ms/Maximum Report Rate)

While this is a requirement for devices that use a bus with high resume latency, it is recommended that solutions that use other buses also implement an input report buffer to avoid interrupt processing glitches that can occur.

Business Justification

Due to host controller or bus resume times from low power states, devices may need to implement buffering to ensure no user interaction data is lost during power transitions.

Device.Input.Digitizer.Pen.ContactReports

Digitizer Reliability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.NoiseSupression**

A digitizer shall only report data corresponding to user input. No other data, often referred to as "Phantom” or “Ghost contacts" shall be reported. This applies both when the system is actively receiving user input and when it is not receiving user input and under both AC and DC power conditions (where applicable).

Device.Input.Digitizer.Pen.CustomGestures

Custom Run-Time System Gestures

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Irrespective of reporting implementation, additional custom run-time pen gestures are prohibited with the exception of wake gestures.

Business Justification

This requirement ensures compatibility with DirectInk.

- Ensures that there is no interference with the core inking experience.
- Ensures compatibility with the inbox gesture engine.

Device.Input.Digitizer.Pen.Eraser

Eraser Affordance

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

An integrated Windows pen solution shall have a hardware affordance on the pen that provides eraser functionality compliant with the associated protocol. The erase functionality must be enabled by default.

Microsoft will allow an exception for active pens that support only a single barrel button until January 01, 2016:

Microsoft strongly recommends that all pens support erase functionality by default. An integrated Windows pen solution that contains support for only a single barrel button will be allowed to ship without an eraser affordance until January 1, 2016. After January 1, 2016, an integrated Windows pen solution shall have a hardware affordance on the pen that provides eraser functionality compliant with the associated protocol and the erase functionality must be enabled by default.

Business Justification

This requirement ensures that interactions with the pen are intuitive and efficient. This implementation enables software developers to develop for eraser functionality while remaining agnostic to the eraser implementation.

Note Active pens that enable eraser functionality provide the most natural means to erase digital ink. Active pens that enable eraser functionality also free the application developer from having to develop UI to switch pen context from ink mode to erase mode. Therefore it is strongly recommended that all pens support erase functionality by default.

Device.Input.Digitizer.Pen.HIDCompliant

HID Compliant Device Firmware and/or HID Mini-port Driver

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.HIDCompliantFirmware**

Applicability: "Windows Desktop"

All digitizers are required to be human interface device (HID) compliant by leveraging an inbox HID mini-port driver and by following the specification for their respective usage in order to report input to Windows.

Applicability: “Windows Mobile”

All digitizers or their associated 3rd party mini-port drivers are required to be human interface device (HID) compliant per the specification for their respective usage in order to report input to Windows.

For more information on implementation, see:

For the reporting requirements, see the collection of topics at http://msdn.microsoft.com/en-us/library/windows/hardware/jj151569(v=vs.85).aspx

For HID over I2C requirements, see http://msdn.microsoft.com/en-us/library/windows/hardware/Dn642101.aspx

For HID over USB requirements, see http://www.usb.org/developers/devclass_docs/HID1_11.pdf

For HID over Bluetooth requirements, see http://developer.bluetooth.org/TechnologyOverview/Documents/HID_SPEC.pdf

Device.Input.Digitizer.Pen.HoverRange

Hover Range

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Digitizer.Pen.PenRange

A pen digitizer shall report the pen location and that it is ‘in range’ starting when the pen tip is at least 5mm away from the screen.

Microsoft reserves the right to enforce hover range >= 10 mm after Oct 1, 2015

Business Justification

This requirement ensures compatibility with DirectInk and more specifically:

- Ensures that users receive on-screen feedback regarding the pen location and tool context in a timely manner.
- Ensures compatibility with Windows palm rejection functionality.

Device.Input.Digitizer.Pen.Jitter

Jitter and Linearity

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Reported jitter from an integrated Windows pen shall be:

- Moving Jitter
  - <= +/- 0.4 mm (center)
  - <= +/- 0.6 mm (within 3.5mm of all edges)
- Stationary Jitter
  - <= +/- 0.3 mm (center)
  - <= +/- 0.5 mm (within 3.5mm of all edges)
- Hover Stationary Jitter <= +/- 0.5mm (center)

Business Justification

This requirement ensures compatibility with DirectInk and more specifically:

- When the pen tip is in contact with the screen and moving, ink accurately reflects actual pen strokes of the user
- When the pen is held stationary (either when in contact with the screen or hovering), jitter is not perceivable to the user.

Device.Input.Digitizer.Pen.Latency

Response Latencies

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Pen digitizers shall have response latencies per the following:

- Cold Boot Latency

- An integrated Windows pen shall be immediately responsive once the display is active
- Cold Boot is defined as the period from application of power to the controller until the controller is ready to accept HID commands

- Down Latency (Idle) <= 150ms (If-Implemented)
  - This requirement is applicable if an idle power state has been implemented
  - Down latency (Idle) is defined as the delay between the pen making initial contact with the screen when the device is in an idle power state and having the associated report delivered to the host.
- Down Latency (Active) <= 35ms (center)
  - Down latency (Active) is defined as the delay between the pen making initial contact with the screen when the device is in an active power state and having the associated report delivered to the host.
- Move Latency <= 30ms (center)
  - While a pen tip is in motion (and in contact with the screen), move latency is defined as the delay between the pen being at a physical location and having the associated report delivered to the host
  - Microsoft reserves the right to enforce move latency <= 15ms (center) after Oct 1, 2015

Business Justification

This requirement ensures:

- Devices are readily available for user interactions after a system power state transition.
- When pen makes initial contact with the screen, lag in visual feedback is not perceivable to most users.
- When pen is in contact with the screen and moving, end-to-end latency of inking and other interactions are not perceivable to most users.

Device.Input.Digitizer.Pen.Pressure

Pressure Reporting

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

Pen digitizers shall report >= 256 levels of pressure.

A logarithmic curve shall be used to report the applied force on the pen tip. When the pen tip is in contact with the screen and the activation force has been exceeded, the device should report the tip switch as set. The maximum activation Force for an integrated Windows pen is <= 20g.

The ideal pressure curve (for a device with 256 levels) is illustrated in Figure 1, however the associated HLK tests will ensure that the device reports pressure within the ranges defined by the upper bound and lower bound. For devices that have greater number of pressure levels, the same curve should be applied but scaled to the number of levels logically reported.

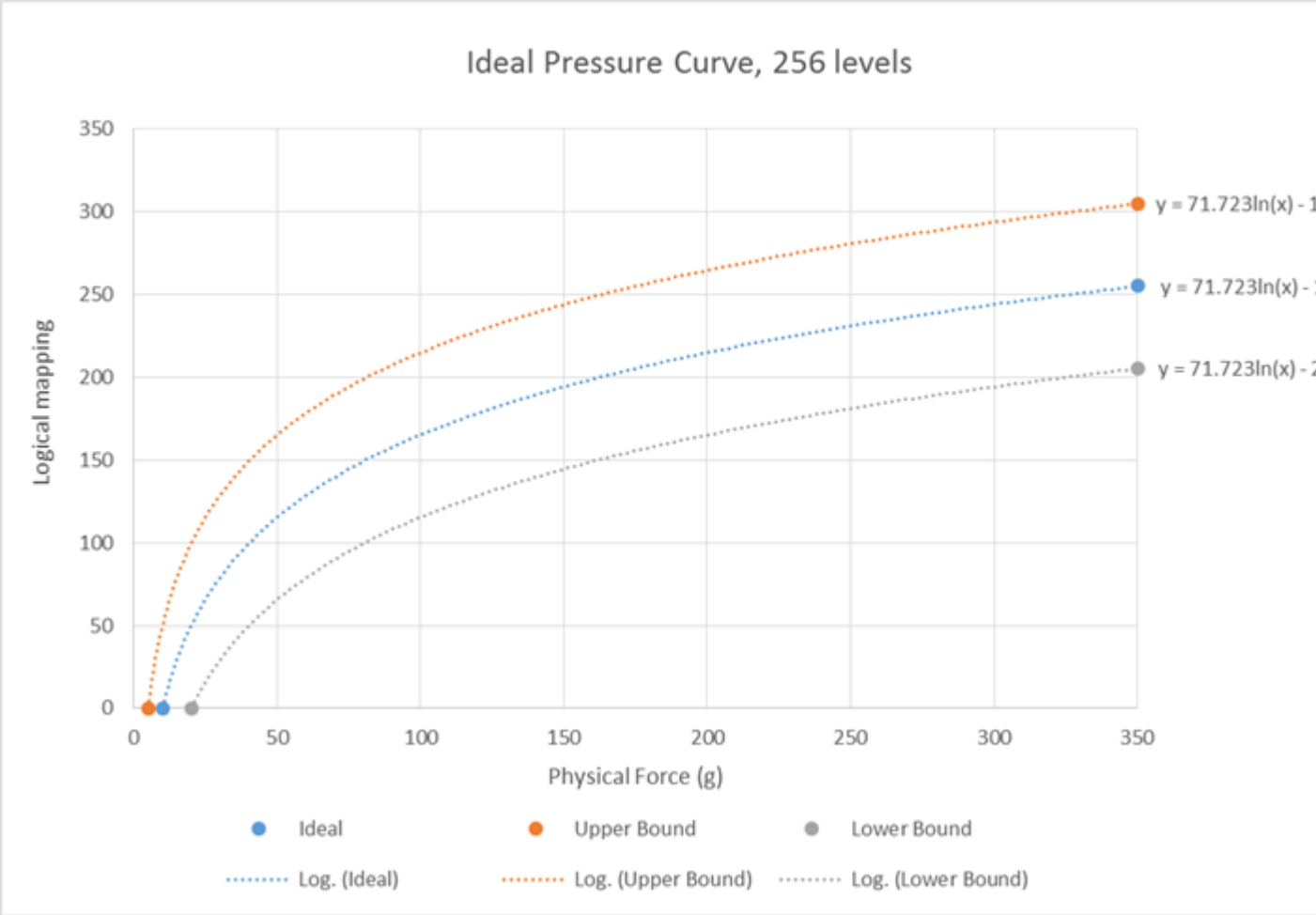


Figure 1 - Ideal Pressure Reporting Curve for a Device with 256 Pressure Levels

Business Justification

This requirement ensures compatibility with DirectInk and more specifically:

- Ensures sufficient granularity and representation for variation in applied pressure as perceived by the user while inking.

Device.Input.Digitizer.Pen.ReportRate

Report Rate

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Digitizer.Pen.100HzSampleRate

An integrated Windows pen digitizer shall have a report rate >= 133Hz.

Business Justification

This requirement ensures compatibility with DirectInk and more specifically:

- Ensures sufficient points are sampled to represent handwriting accurately at typical handwriting velocities.

Device.Input.Digitizer.Pen.Resolution

Input resolution

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Digitizer.Pen.PenResolution

A pen digitizer's input resolution shall be >= the native display resolution or 150 dpi (whichever is greater).

Business Justification

This requirement ensures that every pixel can be inked on screen and provides a smooth pen experience.

Device.Input.Digitizer.Pen.ThirdPartyDrivers

Servicing and 3rd Party Driver Availability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Applicability: Windows 10 for desktop editions (Home, Pro, Enterprise, and Education)

All Windows touch or pen digitizers are required to utilize an inbox HID mini-port driver which functions in absence of any 3rd party HID filter drivers. If a Pen digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM shipping image (and restore image) and available on Windows Update. Any 3rd party HID mini-port driver shall also be functional in WinPE.

Applicability: “Windows Phone”

If a Windows pen digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM Base Support Package (BSP) for the system.

Send comments about this topic to Microsoft

Device.Input.Digitizer.PrecisionTouchpad

In this topic:

- Device.Input.Digitizer.PrecisionTouchpad.SelectiveReporting
- Device.Input.Digitizer.PrecisionTouchpad.ThirdPartyDrivers

Device.Input.Digitizer.PrecisionTouchpad.Accuracy

Accuracy

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows precision touchpads shall report the absolute digitizer position <= 2 mm from actual position for all contacts.

Business Justification

Requirement ensures an accurate moussing and gesturing surface.

Device.Input.Digitizer.PrecisionTouchpad.Buffering

Buffering for buses with high resume latency

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is an “If-Implemented” requirement. USB or Bluetooth connected Windows precision touchpads shall implement an input report buffer capable of handling >= 100 ms of data based on the maximum number of contacts supported by the device Input report buffer size (in bytes):

= Maximum # of Contacts Supported * Input Report Size * (100 ms/Maximum Report Rate)

While this requirement applied to USB and Bluetooth devices, it is recommended that other devices also implement an input report buffer to avoid interrupt processing glitches that can occur.

Business Justification

Due to host controller or bus resume times from low power states, devices may need to implement buffering to ensure that no user interaction data is lost during power transitions.

Device.Input.Digitizer.PrecisionTouchpad.Buttons

Physical buttons and button reporting

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following requirements have been merged into this requirement:

- Device.Input.PrecisionTouchpad.Hardware.ClickpadPress
- Device.Input.PrecisionTouchpad.Hardware.PressurePadPress

If Windows precision touchpads implement external buttons and are not clickpads or pressurepads, there must be at least two buttons to represent left and right click buttons.

If there is more than one button, buttons shall be reported as an array of button flags. Handling buttons beyond the first two (primary and secondary clicks) should be the role of the driver and/or mouse collection.

(If Implemented) – Both clickpad-based and Non-clickpad-based Windows precision touchpads shall report a button down state when exerted pressure exceeds 150g-180g.

Business Justification

Clickable regions should be comfortably usable.

If external buttons are implemented, there must be at least two hardware buttons to represent both left and right click to ensure that the device has full mouse capabilities.

Device.Input.Digitizer.PrecisionTouchpad.ContactReports

Digitizer Reliability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.NoiseSupression**

A digitizer shall only report data corresponding to user input. No other data, often referred to as "Phantom” or “Ghost contacts" shall be reported. This applies both when the system is actively receiving user input and when it is not receiving user input and under both AC and DC power conditions (where applicable).

Device.Input.Digitizer.PrecisionTouchpad.ContactTipSwitchHeight

Contact Tip Switch Height

Windows precision touchpads shall not report contacts > 0.5 mm above the digitizer surface.

Business Justification

Prevents unintended contact activations from occurring and helps to avoid palm invoked activations during typing.

Device.Input.Digitizer.PrecisionTouchpad.DeviceTypeReporting

Device type

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows precision touchpads must report their device button type (e.g. pressure-pad, click-pad) via a HID feature report if a button is present under the digitizer. A device that does not have a clickable digitizer must not expose this feature report.

For more information on implementation, see http://msdn.microsoft.com/en-us/library/windows/hardware/jj126202(v=vs.85).aspx.

Business Justification

Fundamental to ensure the correct button handling is performed by the operating system.

Device.Input.Digitizer.PrecisionTouchpad.Dimensions

Dimensions

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The requirement **Device.Input.PrecisionTouchpad.Hardware.Length** and **Device.Input.PrecisionTouchpad.Hardware.Height** have been merged with this one.

Windows precision touchpads shall be >= 64 mm in length (horizontal measurement of touchpad) x >= 32 mm (vertical measurement).

Business Justification

Precision touchpads must maintain a minimum size to allow users to interact properly with moussing across long distances and performing multi-finger gestures.

Device.Input.Digitizer.PrecisionTouchpad.FingerSeparation

Finger Separation

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Input.PrecisionTouchpad.Performance.MinSeparation.

Windows precision touchpads shall not alias two contacts aligned at a minimum separation of 8mm or less edge-to-edge.

Business Justification

This requirement ensures finger count and position remain accurate during multi-finger interactions.

Device.Input.Digitizer.PrecisionTouchpad.HIDCompliant

HID Compliant Device Firmware and/or HID Mini-port Driver

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.HIDCompliantFirmware**

Applicability: “Windows Desktop” And “Windows Mobile”

All digitizers or their associated 3rd party mini-port drivers are required to be human interface device (HID) compliant per the specification for their respective usage in order to report input to Windows.

For more information on implementation, see:

For the reporting requirements, see the collection of topics at http://msdn.microsoft.com/en-us/library/windows/hardware/jj151569(v=vs.85).aspx

For HID over I2C requirements, see http://msdn.microsoft.com/en-us/library/windows/hardware/Dn642101.aspx

For HID over USB requirements, see http://www.usb.org/developers/devclass_docs/HID1_11.pdf

For HID over Bluetooth requirements, see http://developer.bluetooth.org/TechnologyOverview/Documents/HID_SPEC.pdf

Device.Input.Digitizer.PrecisionTouchpad.InputResolution

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows precision touchpads shall report a logical maximum for x linearly proportional to the width of the touchpad and shall report a logical maximum for y linearly proportional to the length of the touchpad. The top left corner of the touchpad shall be the origin (0,0). All Windows precision touchpads are at least 300 dpi; therefore, for the minimum size Windows precision touchpads, the logical maximum for x shall be >= 767 and the logical maximum for y shall >= 384.

Business Justification

This requirement ensures that users can perform precise manipulations and cursor positioning.

Device.Input.Digitizer.PrecisionTouchpad.Jitter

Jitter and linearity

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The requirement Device.Input.Digitizer.PrecisionTouchpad.Linearity has been merged to this requirement.

One Contact

- Windows precision touchpads shall not report any jitter when a single stationary precision contact is on the digitizer surface.

Multiple Contacts

- Windows precision touchpads shall report contacts with <= 2mm of jitter when multiple stationary contacts are initially placed on the digitizer surface, however no stationary jitter is permitted after the contacts have moved and become stationary again.

Any Number of Contacts

- Windows precision touchpads shall maintain linearity within 0.5mm for all contacts reported across edge to edge travel horizontally, vertically, and diagonally.

- Within 3.5mm of any edge, precision touchpads shall maintain linearity within 1.5mm for all contacts reported.
- Windows precision touchpads shall not report any backward motion jitter for contacts in motion. Backward jitter is defined as a subsequent report of contact location in the opposite direction of contact travel.

Business Justification

Ensures that precision moussing and gestures work correctly and that pausing during gesturing does not result in jittering visuals.

Device.Input.Digitizer.PrecisionTouchpad.Latency

Response latencies

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following requirements have been merged into this requirement:

- Device.Input.PrecisionTouchpad.Performance.ActiveTouchdownLatency
- Device.Input.PrecisionTouchpad.Performance.IdleTouchdownLatency
- Device.Input.PrecisionTouchpad.Performance.PanLatency

Touch down Latency:

- Active:
  - Windows precision touchpads shall have <= 35 ms touch down latency in the active state. Touch down latency is defined as the time between contact with the digitizer and the subsequent input report being received by the host.
- Idle:
  - Windows precision touchpads shall have <= 150 ms touch down latency in the idle state.
  - Touch down latency is defined as the time between contact with the digitizer and the subsequent input report being received by the host.

Move Latency:

Windows precision touchpads shall have panning latency <= 70 ms. Panning latency is defined as the period between the movement on the digitizer and the subsequent change being received by the host.

Cold boot Latency:

Windows precision touchpads shall be immediately responsive once the display is active on systems emerging from cold boot. Cold boot is defined as the period from application of power to the controller until the controller is ready to accept HID commands.

Business Justification

This requirement ensures that core user experiences involving taps, moussing, and gesturing are still performant (i.e. tapping a button, control, hyperlink, etc.).

The touchpad device should be readily available for user input once the system has booted or resumed from a low power state irrespective of its bus connectivity.

Additional UX study on the impact of panning latency for indirect input devices justifies increased panning latency compared to direct input devices.

Device.Input.Digitizer.PrecisionTouchpad.MinMaxContacts

Contact count

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows precision touchpads shall not report being capable of supporting more than a maximum of 5 unique contacts. Windows precision touchpads shall report being capable (and be capable) of supporting a minimum of 3 unique contacts.

Business Justification

Precision touchpads require at least three fingers to have access to all Windows interactions. Four or more fingers are supported for a wider range of available interactions. Multihand interactions are not suited for touchpad use, so more than 5 contacts support is unneeded.

Device.Input.Digitizer.PrecisionTouchpad.ReportRate

Report Rates

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Input.PrecisionTouchpad.Performance.ReportRate.

Windows precision touchpads shall report at >= 125Hz (and no greater than 250Hz) when a single contact is on the digitizer. Windows precision touchpads shall report all contacts at >= Display Refresh Rate + 10Hz (up to 125 Hz) when a multiple contacts are on the digitizer.

Business Justification

Maintain mouse experience requirements and competitive performance at >= 125 Hz. Gesture performance in-line with touch requirements that are backed by the experience needs of Direct Manipulation to avoid stuttering.

Device.Input.Digitizer.PrecisionTouchpad.SelectiveReporting

Selective reporting

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows precision touchpads shall support the hosts request via a HID feature report to selectively report digitizer contacts and button presses. The host may request to receive reports for either digitizer contacts or button presses, neither, or both. Windows precision touchpads shall optimize their power consumption based on host selection.

Business Justification

This requirement ensures that precision touchpad digitizers properly respond to user configuration requests to enable/disable the device.

Device.Input.Digitizer.PrecisionTouchpad.ThirdPartyDrivers

Servicing and 3rd Party Driver Availability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Applicability: "Windows Desktop"

If a Windows precision touchpad digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM shipping image (and restore image) and available on Windows Update. Any 3rd party HID mini-port driver shall also be functional in WinPE.

If a Windows touch or pen digitizer utilizes a 3rd party HID filter driver, it shall be part of the OEM shipping image (and restore image) and available on Windows Update. All Windows touch or pen

digitizers are required to utilize an inbox HID mini-port driver which functions in absence of any 3rd party HID filter drivers.

Applicability: “Windows Mobile”

If a Windows precision touchpad, touch or pen digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM Base Support Package (BSP) for the system.

Send comments about this topic to Microsoft

Device.Input.Digitizer.Touch

In this topic:

Device.Input.Digitizer.Touch.Accuracy

Accuracy

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Description

This requirement was formerly known as Device.Digitizer.Touch.PhysicalInputPosition

A touch digitizer shall report the position of all contacts within the following allowances:

- <= +/- 1mm when contact location is greater than 3.5mm from digitizer edge
- <= +/- 2mm when contact location is within 3.5mm of digitizer edges

Business Justification

This requirement ensures accurate and reliable touch interactions.

Device.Input.Digitizer.Touch.Buffering

Buffering for Buses with High Resume Latency

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This is an “If-Implemented” requirement. A Windows touch solution connected over USB or Bluetooth shall implement an input report buffer capable of handling >= 100ms of data based upon the maximum number of contacts supported by the device

Input report buffer size (in bytes) (for Single Finger Hybrid Reporting): = Maximum # of Contacts Supported * Input Report Size * (100ms/Maximum Report Rate)

While this is a requirement for devices that use a bus with high resume latency, it is recommended that solutions that use other buses also implement an input report buffer to avoid interrupt processing glitches that can occur.

Business Justification

Due to host controller or bus resume times from low power states, devices may need to implement buffering to ensure no user interaction data is lost during power transitions.

Device.Input.Digitizer.Touch.ContactReports

Digitizer Reliability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.NoiseSupression**

A digitizer shall only report data corresponding to user input. No other data, often referred to as "Phantom” or “Ghost contacts" shall be reported. This applies both when the system is actively

receiving user input and when it is not receiving user input and under both AC and DC power conditions (where applicable).

Device.Input.Digitizer.Touch.CustomGestures

Custom run-time system gestures

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Only gestures designed to wake the system are allowed. Custom gestures designed to work with user interface elements are prohibited.

Business Justification

This requirement ensures that the platform maintains a consistent and reliable gesture experience across all Windows devices.

Device.Input.Digitizer.Touch.FingerSeparation

Finger separation

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Digitizer.Touch.InputSeparation

All contacts shall be uniquely tracked and reported when <= 8mm apart measured from edge-to-edge.

Business Justification

This requirement ensures compatibility with inbox gestures such as pinch/zoom, multi-finger taps, and parallel finger manipulations can be performed reliably.

Device.Input.Digitizer.Touch.HIDCompliant

HID Compliant Device Firmware and/or HID Mini-port Driver

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.HIDCompliantFirmware**

Applicability: "Windows Desktop"

All digitizers are required to be human interface device (HID) compliant by leveraging an inbox HID mini-port driver and by following the specification for their respective usage in order to report input to Windows.

Applicability: “Windows Mobile”

All digitizers or their associated 3rd party mini-port drivers are required to be human interface device (HID) compliant per the specification for their respective usage in order to report input to Windows.

For more information on implementation, see:

For the reporting requirements, see the collection of topics at http://msdn.microsoft.com/en-us/library/windows/hardware/jj151569(v=vs.85).aspx

For HID over I2C requirements, see http://msdn.microsoft.com/en-us/library/windows/hardware/Dn642101.aspx

For HID over USB requirements, see http://www.usb.org/developers/devclass_docs/HID1_11.pdf

For HID over Bluetooth requirements, see http://developer.bluetooth.org/TechnologyOverview/Documents/HID_SPEC.pdf

Device.Input.Digitizer.Touch.Jitter

Jitter and linearity

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Digitizer.Touch.DigitizerJitter

Stationary Jitter shall not exceed a maximum of <= +/- 0.5mm away from the edges and <= +/- 1mm within 3.5mm of edges

Moving Jitter shall not exceed a maximum of <= +/- 1mm away from the edges and <= +/- 2mm within 3.5mm of edges

Jitter is not permissible in any direction opposite of contact travel for non-stationary contacts.

Business Justification

This requirement ensures compatibility with inbox gestures, manipulations and inking and more specifically that:

- Tap interactions work with small controls

- Content should not jitter when being held stationary after a manipulation.
- Free-form panning manipulates content in the direction that the user intended
- Finger painting/inking looks acceptable

Device.Input.Digitizer.Touch.Latency

Response latency

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Digitizer.Touch.ResponseLatency

A touch digitizer shall have response latencies as follows:

- Down Latency (Active)<= 35ms
  - Active down latency is defined as the period between contact with the digitizer and the subsequent input report being received by the host.
- Down Latency (Idle) <= 150ms
  - Idle down latency is defined as the period between contact with the digitizer and the subsequent input report being received by the host while the digitizer is connected standby.
- Move Latency

Screen Size (Diagonal)	Maximum Latency
< 7"	<= 35ms
>= 7"	<= 25ms

  - Move latency is defined as the period between a contact being at a physical position and having that position reported to the host.
- Cold Boot Latency -

- Touch digitizers shall be immediately responsive once the display is active. Cold boot is defined as the period from application of power to the controller until the controller is ready to accept HID commands.

Business Justification

This requirement ensures the following:

- Content sticks close to the physical finger position when under manipulation.
- Devices are readily available for user interactions after a system power state transition.
- User interactions involving a tap are responsive even when in a low power state.

Device.Input.Digitizer.Touch.MinContactCount

Minimum simultaneous reportable contacts

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as **Device.Digitizer.Touch.5TouchPointMinimum**.

A touch digitizer shall support a minimum of five simultaneous touch contacts.

Business Justification

This requirement ensures compatibility with inbox gestures, accessibility tools, and 3rd party applications.

Device.Input.Digitizer.Touch.ReportRate

Report rate

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement was formerly known as Device.Digitizer.Touch.ReportingRate

When in an active power state, the touch digitizer reporting rate shall be maintained at a minimum of Display Refresh Rate and a maximum of 250Hz. All reports shall be uniquely sampled.

For example:

Display Refresh Rate	Touch Reporting Rate
60Hz	>= 60Hz
120Hz	>= 120Hz

Business Justification

This requirement ensures a compatible, smooth and stutter-free gesturing experience when performing pan and pinch/zoom interactions utilizing the Windows Direct Manipulation framework.

Device.Input.Digitizer.Touch.Resolution

Input resolution

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

At a minimum, the touch digitizer resolution will be equal the native display resolution or greater. Every pixel of the display in an integrated touch digitizer is required to be accessible to touch input. Every pixel includes pixels on the edges and in the corners of the display.

For additional details on verification and testing of this requirement please see http://go.microsoft.com/fwlink/p/?linkid=234575

Business Justification

This requirement ensures that every pixel is available for touch interaction and ensures a reliable and smooth touch experience.

Device.Input.Digitizer.Touch.ThirdPartyDrivers

Servicing and 3rd Party Driver Availability

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Applicability: "Windows Desktop"

If a Windows precision touchpad digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM shipping image (and restore image) and available on Windows Update. Any 3rd party HID mini-port driver shall also be functional in WinPE.

If a Windows touch or pen digitizer utilizes a 3rd party HID filter driver, it shall be part of the OEM shipping image (and restore image) and available on Windows Update. All Windows touch or pen digitizers are required to utilize an inbox HID mini-port driver which functions in absence of any 3rd party HID filter drivers.

Applicability: “Windows Mobile”

If a Windows precision touchpad, touch or pen digitizer utilizes a 3rd party HID mini-port driver, it shall be part of the OEM Base Support Package (BSP) for the system.

Send comments about this topic to Microsoft

Device.Input.FingerPrintReader

In this topic:

Device.Input.FingerPrintReader.Base

Biometric fingerprint reader general requirements

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Biometric fingerprint readers certified under this program must be exposed through the Windows Biometric Framework (WBF). WBF plug-in components must implement all entry points and adhere to the WBF plug-in interfaces that are documented on MSDN (http://msdn.microsoft.com/library/dd401553(VS.85).aspx).

The following general criteria must be met:

- The driver package must contain a WBDI driver, and a combination of plug-in adapters.

- Devices that operate in the advanced mode may implement a compound adapter. A compound adapter may contain a proprietary sensor, engine, and database adapter, or any combination of these adapters.
- The WBDI driver and plug-in components must not contain any malicious software such as Trojan horse or backdoor software.

The plug-in components must not contain any malicious software such as Trojan horse or backdoor software.

The following table lists the hardware requirements for fingerprint sensors.

Requirement		Justification/Notes
Sensor Type	Touch recommended if implemented	Touch sensor type is the preferred option compare to swipe sensor. Due to the nature, placement, and fixed orientation of swipe sensors especially on devices that can have multiple orientations (such as tablets that can function in landscape and portrait mode), they result in ergonomic challenges for the user which may result in failure to capture an accurate fingerprint thereby resulting in the need for multiple swipes.
Power Consumption	Operating (during capture) 100mW	Required for Connected Standby devices and recommended for others.
	When the fingerprint reader is not capturing fingerprint images regardless of if the system itself is in the sleep state or not - 1mW	Required for Connected Standby devices and recommended for others.
Performance	Recommended <1%FRR @ 0.01% FAR as defined by fingerprint sensor specification	
Liveness Detection	Recommended to be able to provide a minimum of one of the anti-spoofing measures based on the fingerprint sensor specification	

WBF Compliant with HLK	Yes	

Design Notes:

Biometric devices measure an unchanging physical characteristic to uniquely identify an individual. Fingerprints are one of the most common biometric characteristic measured. Beginning with Windows 7, a new set of components, the Windows Biometric Framework, provides support for fingerprint biometric devices. These components, created using the Windows Biometric Framework API, can perform the following tasks:

- Capture biometric samples and use them to create a template.
- Securely save and retrieve biometric templates.
- Map each template to a unique identifier such as a GUID (globally unique identifier) or SID (system identifier).
- Enroll new biometric templates.

For more complete information about WBF and its components, see the "Introduction to the Windows Biometric Framework (WBF)" white paper at http://msdn.microsoft.com/library/windows/hardware/gg463089.aspx.

Device.Input.FingerPrintReader.Extensions

Vendor-supplied drivers or other extension components must not wrap other extension components.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Drivers and adapter plug-ins that extend the Windows Biometric Framework must not wrap other drivers or adapter plug-ins unless the practice is explicitly permitted by the component documentation supplied by Microsoft.

Device.Input.FingerPrintReader.ManagementApps

Biometric fingerprint reader management applications

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

	ARM devices	x86/x64 devices
Enrollment experience (FMA)	Not allowed. An inbox enrollment experience integrated with user settings page is available.	Not recommended. The enrollment experience is provided inbox and integrated with the user settings page. A custom enrollment experience MAY be included with the WBF package and installed on the system however it will not be integrated into user settings page or Control Panel. Custom enrollment experience is not allowed to enroll non-domain users.

Exceptions

Devices targeted for consumer use MUST not have any 3rd party FMA installed. 3rd party FMAs may be installed and enabled on Windows devices targeted towards business (enterprise and small/medium businesses). In such instances, the user should be clearly notified if core inbox experiences will be negatively impacted by the use of the 3rd party FMA during enrollment.

Device.Input.FingerPrintReader.SensorEngineDB

Fingerprint reader sensor, engine and storage plug-in requirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- All interface entry points must be implemented, and each element must adhere to the definition of its respective adapter plug-in interface.
- Plug-in components must support the sequenced invocation of their interfaces that result from invoking all the top-level Windows Biometric Framework (WBF) APIs.
- The sequence of WBF API calls used by the WBF credential provider for Windows logon must be completed in 1.5 seconds or less.

- All adapters must be digitally signed, as described in the Windows Biometric Framework: Code-Signing Guidelines white paper at http://msdn.microsoft.com/library/windows/hardware/gg463093.aspx.
- The adapter must run under Application Verifier without generating any errors.

A driver package can support multiple devices, but must pass the certification criteria for each of the supported fingerprint readers separately. A separate submission must be made for each supported device.

Adapter requirements	Notes
Engine adapter	Required unless the device is an advanced device that can match on device.
Storage adapter	Relying on inbox storage adapter is recommended unless there is a specific need to include a custom storage adapter such as for advanced sensors.
Sensor adapter	Relying on inbox sensor adapter is recommended unless there is a specific need to include a custom sensor adapter.

Device.Input.FingerPrintReader.WBDI

Biometric fingerprint reader WBDI driver requirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows Biometric Driver Interface (WBDI) drivers must adhere to the following criteria:

- All mandatory IOCTLs must be fully implemented and adhere to the WBDI specification.
- If an optional IOCTL is implemented, it must adhere to the WBDI specification.
- The WBDI driver must support the biometric IOCTL calling sequence.
- If the biometric fingerprint reader is not a PnP device, it must create arrival and departure events as recommended in the WBDI specification.

- The WBDI INF installer must set up the associated plug-in components and fingerprint management applications that are included in the driver package.

Guidelines in this requirement apply equally well to the kernel-mode and user-mode drivers. The following must be supported:

- Backward and forward compatibility. Drivers must have a mechanism that determines different versions of user-mode and kernel-mode components. When two versions of the same driver have been installed on the PC, the kernel component must determine the correct driver to talk to. In remoting scenarios, the mismatch may occur even with a single driver.

The following items must not be used:

- Out-of-band communications. Because the user-mode and kernel-mode drivers are running on the same PC, they might be coupled together through channels different from the I/O manager APIs. The goal is to ensure that drivers do not have out-of-band communication, implicit or explicit.

Here are some specific examples that would prevent terminal services redirection:

- Shared memory must not be used directly between user-mode applications and either user-mode or kernel-mode drivers. For every data item sent and received, there must be a corresponding I/O request. Share the memory either through a mapped file or through locked pages of one direct I/O call.

- Registry communication (that is, any registry keys that are set from user-mode drivers and read from kernel-mode drivers or vice versa). It is problematic when an application is running on the server and drivers are loaded on the client because the registry is set on the server, but not on the client.

- Kernel objects (passing any kernel object handles in I/O buffers, such as handles to events or semaphores).

- Data pointers (passing data pointers in I/O buffers). For example, a driver may try to read a linked list or other complicated data structure from the kernel.

- Incompatibility between 32-bit and 64-bit platform implementations of the I/O request packet (IRP) requests (fields in the data structures that are compiled differently, depending on the bitness). Incompatibility of the structures based on bitness results in different offsets and data size for these structures. The data will not be interpreted correctly when a terminal services client and server are not the same platform.

- Reliance on a strict timing expectation about how fast the kernel driver responds. Because drivers are remoting over the network, additional latency is added to the response from the hardware. That latency may range from 10 ms to serveral seconds. A possible cause for this could be slow network or packet losses. If a driver is programmed for a response that comes in time less than usual network latency, the remoting fails.
- Setting an access control list (ACL) or any other run-time security check that would prevent any application from opening a device driver. For example, a driver is allowed to accept calls only from particular service. Because all the calls are done on a TS client PC under security context of the remoting client process, they can fail.

Design Notes Biometric devices measure an unchanging physical characteristic to uniquely identify an individual. Fingerprints are the most common biometric characteristic measured. Beginning with Windows 7, a new set of components, the Windows Biometric Framework (WBF), provides support for fingerprint biometric devices. These components, created with the WBF API can perform the following tasks:

- Capture biometric samples and uses them to create a template.
- Securely save and retrieve biometric templates.
- Map each template to a unique identifier such as a GUID (globally unique identifier) or SID (system identifier).
- Enroll new biometric templates.
- To expose a device through WBF, the device must be exposed through a driver that implements the WBDI driver as well as an appropriate combination of sensor, engine, and database plug-in components. For more complete information about WBF and its components, see the "Introduction to the Windows Biometric Framework (WBF)" white paper at http://msdn.microsoft.com/library/windows/hardware/gg463089.aspx.

Send comments about this topic to Microsoft

Device.Input.HID

In this topic:

Device.Input.HID.I2CProtocolSpecCompliant

All HID devices connected over I2C must comply with Microsoft HID I2C Protocol specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All HID over I2C compatible devices must comply with the Microsoft published documentation for the HID I2C protocol specification.

Design Notes:

See Microsoft published HID I2C protocol specification (link not provided yet)

Exceptions:

This requirement is only enforced for HID I2C devices and not generalized for SPB.

Device.Input.HID.UsbSpecificationCompliant

All HID devices that are connected over USB must comply with the USB HID Specification (V1.1 or later).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Version 5: Defines WMC AQ requirements

- For WMC AQ, this requirement is:
  - REQUIRED for desktop systems
  - REQUIRED for laptop systems if laptop system includes remote and receiver.
- For non AQ systems, this requirement is:
  - REQUIRED if system (either desktop or laptop) includes remote and receiver.
- IR receivers (input only) and IR transceivers (input and IR emitting/learning) interface with the Windows Media Center class driver. In a system with a TV tuner, IR Transceivers must be used that support all functions of the Remote Control and Receiver-Transceiver

Specifications and Requirements for Windows Media Center in Windows Operating Systems document.

- For tunerless systems with an IR receiver, only IR Input and wake functions are required. The IR learning and emitting functions are optional.
- DVB-T solutions are not required to support learning and emitting. These functions are optional.
- In locales that do not use set top boxes, the learning and emitting functions are optional.
- In those regions that support a secondary 10 foot application, the IR receiver/transceivers are not required to meet the IR learning requirement.
- In those regions that support DVB-S, Microsoft strongly recommends the use of IR transceivers (input and IR emitting/learning).
- If shipping a laptop system, IR learning and IR emitting is optional.
- For IR receivers (input only) that use Microsoft authorized IR protocols and all IR Transceiver (input and emitter functions), the device will return an IR waveform envelope to software for software decoding of IR signal. An IR signal cannot be decoded in the hardware. The only exception to this is the "wake-from-remote" power key.
- An IR receiver (input only) is allowed to perform hardware decoding of an IR signal. These IR receivers (input only) must not receive and respond to any currently authorized Microsoft IR protocol. IR receivers that use hardware decoding of an IR signal also need to support the "wake-from-remote" functionality. These devices must comply with the Remote Control and Receiver-Transceiver Specifications and Requirements for Windows Media Center in Windows Operating Systems document.

Design Notes:

Microsoft recommends that IR cables be labeled and well documented for end users. An insert showing a small diagram of the IR control cable and how it connects to the digital cable or satellite receiver could help prevent support calls.

Send comments about this topic to Microsoft

Device.Input.Keyboard

Logo requirements detailing the implementation details of a keyboard important to Microsoft operating systems

In this topic:

Device.Input.Keyboard.BrowserMultimediaKeysUseMSApis

Keys for Internet browser and multimedia must use Microsoft APIs.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a keyboard or peripheral implements multimedia or Internet browser keys, it must use the registry keys associated with the WM_APPCOMMAND API to access those functions as described in the Windows Driver Kit. Registry keys can be programmed by using INF files to install special entries as defaults or through a customized interface provided to the user.

Design Notes:

See the Microsoft Platform SDK, "WM_APPCOMMAND."

Device.Input.Keyboard.CharmsKey

If any of the Windows Charms keys are implemented on keyboards, then it must implement the correct scan codes and proper glypths.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Keyboards that implement buttons to launch any of the Windows Charms must use the correct scan codes and glyphs on those buttons. The glyphs that go on the buttons are defined in the Windows Glyphs addendum to the Windows Logo License agreement for Hardware available at http://go.microsoft.com/fwlink/?LinkId=279574.

The charm button must send the correct scan code corresponding to the charm. No other glyph can be used on the button when using a scan code that relates to invoking one of the Windows 8 charms.

Device.Input.Keyboard.DynamicKeyboards

Dynamic keyboards must meet the requirements listed here.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- When a system is displaying the security desktop, a keyboard capable of altering the keycaps to reflect different glyphs or legends dynamically must present a keyboard layout to match the language the current user has active on the system.
- When using a keyboard capable of altering the keycaps to reflect different glyphs or legends dynamically, the keyboards must reboot into the default system language layout as selected in Control Panel -> Regional and Language Options -> Keyboards and Languages.
- When using a keyboard capable of altering the keycaps to reflect different glyphs or legends dynamically, the keyboards must change the keyboard layout and language as selected by a user at the login screen.
- When using a keyboard capable of altering the keycaps to reflect different glyphs or legends dynamically, the keyboards must reflect the currently selected layout and language preference when the Windows desktop has focus.
- When using a keyboard capable of altering the keycaps to reflect different glyphs or legends dynamically, the keyboard may allow for the repositioning of the Windows key; however, that key must remain visible to user in all configurations/layouts. By default, this key must be present in the lower left of the keyboard, between the control and alternate keys when at a login, welcome, or password entry screen. Once the user has logged in, the location of the key may be repositioned by user preference.

- When using a keyboard capable of altering the keycaps to reflect different glyphs or legends dynamically, the displayed keyboard layout and language must match the installed language of the operating system.
- A self-powered keyboard capable of altering the keycaps to reflect different glyphs or legends dynamically, must allow for the keyboard to be reset via a switch or keystroke sequence, independently of a software reset or power cycling of the device.

Device.Input.Keyboard.HotKeyFunctionAPI

Devices that implement Hot-Key functionality must implement the corresponding API notifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Pointing and drawing devices and keyboards, that implement buttons used as application or function hot keys for which there exists WM_APPCOMMAND API shall implement the corresponding API notification.

This includes, but is not limited to the following functions:

- Volume up/down
- Mute
- Browser back/forward
- Play/pause

Design Notes:

Best practices for supporting pointing and drawing devices and keyboards can be found at http://msdn.microsoft.com/en-us/library/ms997498.aspx.

API for WM_APPCOMMAND notifications can be found at http://msdn.microsoft.com/en-us/library/ms646275(VS.85).aspx.

HID Usage Page and ID information for these functions can be found at http://www.microsoft.com/whdc/archive/scancode.mspx and http://www.usb.org/developers/hidpage.

Device.Input.Keyboard.KernelModeDriversUseWdfKmdf

Keyboard kernel-mode drivers must use the WDF-KMDF.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Third-party keyboard kernel-mode drivers must be ported to the WDF KMDF model.

Device.Input.Keyboard.LogoFlagKey

Windows symbol key is implemented on all keyboards supporting more than 50 keys. The Windows symbol design is required after December 31st, 2013.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All keyboards that support more than 50 keys must implement the Windows Symbol key. The printed Windows flag logo version of the key design may be logo qualified on mobile systems and standalone keyboards until the transition date. The Windows flag trademark must be clearly distinguished on the key top according to The Microsoft Windows Logo Key Logo License Agreement and the "Key Support, Keyboard Scan Codes, and Windows" document at http://go.microsoft.com/fwlink/?LinkId=116451.

Device.Input.Keyboard.MultipleKeyboard

No interference occurs between multiple keyboards.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the system includes more than one keyboard, there must be no conflicts. For example, a docked mobile computer can have more than one keyboard attached to the system. The keyboard ports on a mobile computer and a docking station must be able to resolve conflicts between the two ports when the mobile computer is docked.

Device.Input.Keyboard.ScanCode

Scan codes must comply with industry standards.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following are requirements for a keyboard design that includes any Windows logo keys:

- The keyboard must be developed according to technical requirements in "Key Support, Keyboard Scan Codes, and Windows."
- The keyboard must be compatible at the Windows virtual key-code level.
- The Windows logo key must function as a modifier (CTRL, SHIFT, or ALT).
- Keyboard manufacturers must use consumer control or vendor-specific, top-level collections for HID hot buttons.

Design Notes:

See "Key Support, Keyboard Scan Codes, and Windows" at http://go.microsoft.com/fwlink/?LinkId=36767.

Additional software or drivers can be written to provide software remapping functionality.

Send comments about this topic to Microsoft

Device.Input.Location

Windows 10 location drivers can implement either of the following driver models.

- GNSS drivers integrating with the legacy sensor class extension driver

OR

- GNSS driver integrating with the GNSS DDI introduced in Windows 10

In this topic:

Device.Input.Location.AssistedGNSS [If Implemented]

Only relevant if the hardware uses the location stack for Assisted GNSS of the given type.

Applies to	Windows 10 x64 Windows 10 x86

Description

Assistance to the GNSS receiver is used to enable the receiver to acquire a position faster, to enable acquisition even in cases of very weak signal, and do so at lower power. The Location platform supports providing the GNSS device with a coarse position, approximate time, and it can also serve as a proxy of the proprietary ephemeris formats supported by the GNSS device.

Device.Input.Location.Base

Location devices must support the following basic functionality.

Applies to	Windows 10 x64 Windows 10 x86

Description

Location devices must be able to interact correctly with the platform to receive location requests and provide a response with a position or an error. The exchange of capabilities must supported, as well as the configuration commands, for whichever functionality is present and mandatory in the driver interface. All location devices need to ensure that they report location data at an interval determined by the type of location session and the location session parameters. The application desired accuracy and other session parameters should be considered to provide location data as efficiently as possible.

As two driver models are supported, we have requirements specific requirements and tests for each.

For GNSS drivers integrating with the legacy sensor class extension driver

The GNSS device must support the following basic requirements:

- **Data Field Support for Location Reports**

  - The Location API exposes location data through standard built-in reports. These reports are populated from location sensors that the Location API manages automatically.

  - Location sensor devices that report their category as SENSOR_CATEGORY_LOCATION (though Device Driver Interfaces ISensorDriver::OnGetSupportedSensorObjects and IsensorDriver::OnGetProperties()) must support one of the built-in reports so that the location API can manage the sensor and report its data. Each built-in report has a set of expected data fields (though Device Driver Interface

IsensorDriver::OnGetSupportedDataFields() and IsensorDriver::OnGetDataFields()) that the Location API looks for.

- All built-in reports require SENSOR_DATA_TYPE_TIMESTAMP to be provided in addition to the designated fields. This field reports the time the data was collected in UTC.

- The Location API supports a built-in report: the LatLongReport, which requires in the response measurements for the Latitude/Longitude (lat/long) and an Error. Support for the LatLongReport is required for all location sensors.

To support LatLongReport, the following fields are required:

Data field	Data type	Details
SENSOR_DATA_TYPE_LATITUDE_DEGREES	VT_R8	Shows the current latitude in degrees, which must be in the range [-90, +90]
SENSOR_DATA_TYPE_LONGITUDE_DEGREES	VT_R8	Shows the current longitude in degrees, which must be in the range [-180, +180]
SENSOR_DATA_TYPE_ERROR_RADIUS_METERS	VT_R8	The actual latitude and longitude must be within a circle, with this radius (in meters) drawn around the reported (latitude, longitude). Enables the Location API to prioritize sensors; it should be updated dynamically and must be non-zero and positive.

As part of the LatLongReport, the following fields are optional, but must be reported when available:

Optional Data field	Data type	Details
SENSOR_DATA_TYPE_ALTITUDE_ELLIPSOID_METERS	VT_R8	The altitude with regards to the WGS84 ellipsoid (in meters)

SENSOR_DATA_TYPE_ALTITUDE_ELLIPSOID_ERROR_METERS	VT_R8	The error of the current altitude measurement (in meters)

Important: The Location API supports an additional set of data fields (which correspond to NMEA data fields). For more information, see the Sensors topic in the Device and Driver Technologies section of the WDK.

The requirement below ensures that the driver raises location report events in the correct manner. If the correct behavior is not followed, the Location API will block the sensor, and client applications will not receive data.

- **Generating Sensor Data Reports**

  The device must be able to generate a series of valid ISensorDataReports for one of or both of the two Location Reports types. Each report generated must contain at least the minimum data fields for the report type. The Sensor and Location Platform will rely on data provided to the platform via device drivers that have implemented the Sensor Device Driver Interface. In most cases devices will be verified by using the Sensor and Location Platform.

- **Notes about settable properties**

  The following settable properties must be utilized in the device driver as filtering and power management criteria. Both properties must be tracked on a per-client basis.

  The properties are required to be implemented as settable:

Property	Data type	Details
SENSOR_PROPERTY_CURRENT_REPORT_INTERVAL	VT_UI4	Sets the minimum frequency (in milliseconds) that a client wants to receive data reports from the sensor.
SENSOR_PROPERTY_LOCATION_DESIRED_ACCURACY	VT_UI4	Sets a hint as to how accurate the driver should strive to be.

		DESIRED_ACCURACY_DEFAULT: The sensor should optimize power and other cost considerations. GNS sensors will not be enumerated by the Location API unless location data is unavailable from other providers on the system or data accuracy is less than 500m. DESIRED_ACCURACY_HIGH: The sensor should deliver the highest accuracy report possible. The Location API will always connect to all location sensors (including GNSS) in order to acquire the most accurate position possible.

For GNSS driver integrating with the GNSS DDI introduced in Windows 10

The GNSS device must support the following basic requirements:

Platform and Driver Capability Exchange	Check for the exchange between the platform (GNSS_SEND_PLATFORM_CAPABILITY) and the driver (GNSS_GET_DEVICE_CAPABILITY) regarding their respective capabilities. The expectation is that the driver correctly sends to the platform its device capabilities, given that the location platform will adapt its behavior accordingly. Capabilities such as distance tracking, and geofencing shall only be declared when they can be offloaded to the GNSS device, this is, when they do not require the application processor to do the tracking.
Driver Command check	Check that the basic driver commands that are required for both product and test enablement. Like, ClearAgnssData.
Basic Single Shot check	Begins a single shot session and runs until a final fix is received. Expect that a final fix is delivered within 2 minutes. Assistance data could be available in this case. In case assistance data is available a final fix is expected to be delivered in 10 seconds.
Totally standalone basic single shot check	Ensures that a driver that is in a ‘cold state’, and does not have access to AGNSS of any type, can still achieve a final fix for a single shot request. Expect that a final fix is delivered within 2 minutes.

Single Shot Multiple Adjacent Session check	Checks that one hundred single shot sessions can be run one after another without any issues. Expect that every single shot session reaches a final fix without issue.

Device.Input.Location.Geofencing [If Implemented]

This requirement is only supported if the hardware supports Geofencing and the GNSS driver is integrating with the GNSS DDI introduced in Windows 10.

Applies to	Windows 10 x64 Windows 10 x86

Description

This requirement allows the GPS to define geographical boundaries or virtual barriers. It allows an administrator of a mobile device to trigger actions or notifications when a device enters or leaves the geofence. Practical uses include the ability for a retail store owner to create a geofence around the retail store and send coupons to a customer with the retail store app on his/her mobile device.

If implemented, the device must be able to support the following requirements.

Geofence basic	GeoFence can be successfully added and deleted. Must check for consistency.
Geofence track 1	Add a Geofence around your current position with UNKNOWN state and verify ENTRY event.
Geofence track 2	Add a Geofence with ENTERED state such that your current position is just outside the Geofence. Verify Exit event.
Geofence track 3	Add a Geofence around your current position with ENTERED state and verify that no event is triggered
Geofence track 4	Add a Geofence with EXITED state such that your current position is just outside the Geofence. Verify no event is triggered.
Geofence track 5	Add a Geofence around your current position with UNKNOWN state and Alert Type as EXIT only and verify that no event is recieved
Geofence track 6	Add a Geofence (with Entered state) such that your current position is just outside the Geofence and set the Alert type to ENTER and verify that NO event is recieved

Reliability	Obtains the maximum number of geofences that the hardware supports. Set this maximum number of Geofences around current position with unknow state and expect that number of entry/exit events.
Geofence purge	Add multiple Geofences and then purge them all at once. Must check for consistency.
Geofence negative	Add a Geofence with bad (Zero radius) parameters.
Geofence boundary	Add a Geofence with very small radius (5m).

Device.Input.Location.SUPL [If Implemented]

This requirement is applicable only to a GNSS driver integrating with the GNSS DDI introduced in Windows 10.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Secure User Plane Location (SUPL) client must be implemented by the IHV to support Mobile Operator requirements. The driver interface provides mechanisms to configure the SUPL client, with the information obtained for the Mobile Operator. The only SUPL configurations supported by the Location Platform are those associated to a Mobile Operator, and only one SUPL configuration is supported at a time.

The tests in this section ensures that the SUPL client is able to receive SUPL configuration commands from the Location Platform. There are no tests in the HLK to validate the integration between the GNSS driver and the Location Platform for the support of user notifications upon reception of NI (Network Initiated) location requests. Such validations will need to be done using dedicated systems and test suites that validate SUPL OMA conformance.

Device.Input.Location.V2PUL [If Implemented]

This requirement is applicable only to a GNSS driver integrating with the GNSS DDI introduced in Windows 10.

Applies to	Windows 10 x64 Windows 10 x86

Description

The V2 User Plane Location (V2UPL) requirement applies only to devices with CDMA cellular connections, and if required by the Mobile Operator.

Send comments about this topic to Microsoft

Device.Input.PointDraw

Applies to mice, touch pads, and other input devices used to move the pointer

In this topic:

Device.Input.PointDraw.KernelModeDriversUseWdfKmdf

Mouse kernel-mode drivers must use the WDF-KMDF.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Third-party mouse kernel-mode drivers must be ported to the WDF KMDF model.

Send comments about this topic to Microsoft

Device.Input.SmartCardMiniDriver

MiniDriver program for smart cards

In this topic:

Device.Input.SmartCardMiniDriver.DoNotStopWhenResourcesAreUnavailable

Smart card driver must not stop the system if required resources are not available.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A smart card driver must not interrupt system operation if resources that are required by the reader are not available.

Device.Input.SmartCardMiniDriver.SpecsAndCertifications

Windows Smart Card Minidrivers must meet Windows Smart Card Minidriver Version 5 Specifications and Certification Criteria.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Smart Card Minidrivers are pluggable security components that provide an abstraction layer between the base CSP and the smartcard to provide secure storage for cryptographic keys and certificates. Smart Card Minidrivers perform secure cryptographic operations including encryption, decryption, key establishment, key exchange, and digital signatures. Smart Card Minidrivers also include other form factors, such as a USB tokens or other personal trusted devices.

Smart Card Minidrivers must adhere to the following specifications:

- Smart Card Minidriver Specification for Windows Base Cryptographic Service Provider (Base CSP) and Smart Card Key Storage Provider (KSP), Version 5.06 (or later).
- Smart Card Minidriver Certification Criteria, Version 5.06 (or later).

Smart Card Minidrivers must adhere to the following basic criteria:

- If the device submitted for testing is a smart card and has an ISO 7816 ID-1 smart card form factor, it must be tested with a smart card reader that has passed the WHQL Testing Requirements for smart cards.
- If the device is a multi-function device, it must pass the compatibility requirements for each device category if a compatibility program exists.
- The card minidriver may not implement additional functionality beyond that specified in the Card Minidriver Specification.

- The card minidriver may not contain any Trojan's or "backdoors".
- The card minidriver may not present any UI to the end user.
- All cryptographic operations must take place on the device.
- All cryptographic keys must be stored on the device and must not be exportable from the device.

Smart Card Minidrivers must adhere to the following general criteria in the Card Minidriver Certification Criteria:

- Card Minidriver Management and Installation
- Card Minidriver Logical File System Requirements
- Card Minidriver General Conventions
- Card Minidriver Memory Management
- Optional General Requirements

Smart Card Minidriver must adhere to the criteria governing each of the Functional Exports for each function in the Card Minidriver specification, including:

- Functionality
- Performance
- Error Handling

Smart Card Minidrivers must support the sequenced invocation of card minidriver functions.

A Smart Card Minidriver may support multiple smart cards, but must pass the certification criteria for each of the supported smart cards separately.

Design Notes:

See Smart Card Minidriver Specification for Windows Base Cryptographic Service Provider (Base CSP) and Smart Card Key Storage Provider (KSP), Version specifications at http://msdn.microsoft.com/en-us/windows/hardware/gg487500.aspx.

See Smart Card Minidriver Certification Criteria, at http://msdn.microsoft.com/en-us/windows/hardware/gg487504.

The following table describes the minimum and maximum specification version that must be supported on any given OS family:

Operating system family	Allowed specification versions

Windows 7 client	5,6,7 (any combination allowed such as 5 and 7 only, 5 only, 7 only, 5 and 6 only, 6 and 7 only etc.)
Windows Server 2008	5,6,7 (any combination allowed such as 5 and 7 only, 5 only, 7 only, 5 and 6 only, 6 and 7 only etc.)
Windows 8 client	6,7,8 (any combination allowed such as 6 and 8 only, 6 only, 7 only, 6 and 6 only, 7 and 8 only etc.)
Windows Server Technical Preview	6,7,8 (any combination allowed such as 6 and 8 only, 6 only, 7 only, 6 and 6 only, 7 and 8 only etc.)

Device.Input.SmartCardMiniDriver.SupportMultipleInstancesOnASystem

Smart card driver can support multiple instances of the same device on a system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The smart card driver must be able to function properly if more than one instance of the devices is installed on a system. The functionality of each device instance must be consistent. When a separate instance is loaded, functionality cannot be reduced.

Send comments about this topic to Microsoft

Device.Input.SmartCardReader

In this topic:

- Device.Input.SmartCardReader.SupportsMinClockFrequency
- Device.Input.SmartCardReader.SupportsMinDataRateOf9600bps
- Device.Input.SmartCardReader.SupportsNegotiableAndSpecificModes
- Device.Input.SmartCardReader.SupportsResetCommand
- Device.Input.SmartCardReader.UsbCcidCompliesWithUsbDeviceClassSpec
- Device.Input.SmartCardReader.UsbCcidIssuesNak

Device.Input.SmartCardReader.PinDataEntryKeyboardCompliesWithIso

An input device that implements a PIN data-entry keyboard must comply with ISO.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

An input device that uses a keyboard for PIN entry must comply with ISO 13491-1:1998 Banking--Secure Cryptographic Devices (retail) Part 1: Concepts, Requirements, and Evaluation Methods.

Device.Input.SmartCardReader.SmartCardService

The Smart Card Service must start after a Smart Card is inserted into a reader.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The Smart Card Service must be started after a Smart Card is inserted into the Smart Card reader.

Business Justification

This requirement is necessary for reliability of the smart card function.

Device.Input.SmartCardReader.Supports258And259BytePackets

A reader must support 258-byte packets in T=0 and 259-byte packets in T=1.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

A smart card reader must support the exchange of the following in a single transmission:

- .258-byte packets in T=0; that is, 256 data bytes plus the two status words SW1 and SW2.
- 259-byte packets in T=1; that is, 254 information bytes plus node address, packet control bytes, length, and two error detection code bytes.

Device.Input.SmartCardReader.SupportsDirectAndInverseConvention

A smart card reader must support direct and inverse-convention smart cards.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A smart card reader must support both direct and inverse-convention smart cards either in hardware or in the operating system driver.

Device.Input.SmartCardReader.SupportsInsertionAndRemovalMonitor

A reader must support smart card insertion and removal monitor.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A smart card reader must be able to detect and report smart card insertions and removals with no user intervention other than removing or inserting the smart card itself. The reader must use an interrupt mechanism to report the smart card insertion or removal to the system. A driver polling method to detect smart card insertion and removals is not an acceptable way to meet this requirement.

Device.Input.SmartCardReader.SupportsMinClockFrequency

A smart card reader must support a 3.5795-MHz minimum clock frequency.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A smart card reader must support a minimum clock frequency of 3.5795 MHz.

Device.Input.SmartCardReader.SupportsMinDataRateOf9600bps

A smar card reader must support a minimum data rate of 9600 bps.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A smart card reader must be able to transfer data at a rate of 9600 bps or higher.

Device.Input.SmartCardReader.SupportsNegotiableAndSpecificModes

A reader must support negotiable and specific modes according to the ISO/IEC specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To support multiple-protocol smart cards and smart cards that use higher data rates and higher clock frequencies, the reader must support negotiable and specific modes according to ISO/IEC 7816-3 (1997-12-15), Sections 5.4 and 7.

The **Power Down** command for ISO 7816-3 is optional, but the **Reset** command is required.

PTS is not required.

Device.Input.SmartCardReader.SupportsResetCommand

A smart card reader must support the Reset command.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A smart card reader must support the asynchronous protocols T=0 and T=1 as described in either the hardware or the driver. Both protocols must be fully supported. The smart card reader and the driver must support cards that can handle both protocols. Support is not required for protocols other than T=0 and T=1.

The following protocol rules apply for the T=1 protocol:

- A transmission is defined as sending a command to a smart card by using one or more T=1 blocks and receiving the corresponding answer by using one or more T=1 blocks as defined in ISO/IEC 7816-3.
- For cards that support IFSC requests, the first transmission after a reset of the smart card must begin with an IFSD request, as defined in ISO/IEC 7816-3, Amendment 1, Section 9.5.1.2.
- For cards that do not support an IFSD request (that is, the card replies with an R-Block indicating "Other error"), the transmission must continue with an I-Block.
- After a successful RESYNCH request, the transmission must restart from the beginning with the first block with which the transmission originally started.

Device.Input.SmartCardReader.UsbCcidCompliesWithUsbDeviceClassSpec

A USB smart card CCID reader must comply with USB Device Class Specification for USB Chip/Smart Card Interface Devices.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the reader supports USB connectivity, CCID is required. To ensure that USB smart card readers function properly with the USB host, smart card CCID readers must comply with USB Device Class: Smart Card Specification for Integrated Circuit(s) Cards Interface Devices, Revision 1.00 or later.

Device.Input.SmartCardReader.UsbCcidIssuesNak

A USB CCID reader must issue an NAK on an interrupt pipe if a device has no interrupt data to transmit.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB CCIDs must issue an NAK on an interrupt pipe unless the state changes. This prevents the necessity of repeatedly polling the device for status.

Design Notes:

See USB Device Class Specification for USB Chip/Smart Card Interface Devices, Revision 1.00 or later, Chapter 3. See USB Specification, Revision 1.1 or later, Sections 5.7.4 and 8.5.4.

Send comments about this topic to Microsoft

Device.Network.DevFund

Network requirements

In this topic:

Device.Network.DevFund.NdisVersion

NDIS devices must conform to the NDIS 6.x requirements in the Windows Driver Kit.

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

All NDIS device must conform to NDIS 6.x specified in the Windows Driver Kit.

Design Notes:

See the Windows Driver Kit, "NDIS."

Device.Network.DevFund.NPOS

Network Devices must support No Pause On Suspend (NPOS).

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

NDIS miniport drivers must support No Pause On Suspend (NPOS) on client SKUs (feature support on server SKUs is optional).

Design Notes:

See the No Pause On Suspend Specification.

Device.Network.DevFund.SelectiveSuspend

NDIS devices must meet Selective Suspend requirements.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

NDIS devices must meet Selective Suspend requirements.

Send comments about this topic to Microsoft

Device.Network.LAN

In this topic:

Device.Network.LAN.CloudStress

Ethernet Devices that implement GRE Encapsulated Packet Task Offloads must comply with the specification.

Applies to	Windows Server 2016 Technical Preview x64

Description

This requirement applies to all Ethernet devices that are Certified for Server. This test validates that an Ethernet device can meet the stress requirements of a datacenter cloud. Ethernet devices that meet this requirement must also comply with the specification and support the following features:

All Network Adapters must support the following Devie.Networking.LAN features and pass the Private Cloud Stress test

Base (requirements 100MbOrGreater through SupportIEEE8023)

ChecksumOffload

LargeSendOffload

RSS

Network Adapters for small Cloud Ready deployments must support all of the above features and

Have speeds of 10G or higher

MTUSize

NVGRE

VXLAN

VMQ

Network Adapters for large Cloud Ready deployments must support all of the above features and

PacketDirect

DCB

KRDMA (iWARP or Routable RoCE)

SRIOV

Additional Information;

1GigE NICs, if they do NOT support all the required features will NOT be granted Logo for server

1GigE NICs, if they support all the required features will be granted Logo for server

1GigE NICs, even if they support some of the advanced features listed for Cloud Ready NICs, will not be granted Cloud Ready Additional Qualification

10GigE and above NICs, if they do NOT support all the required features, regardless of any additional features they support, will NOT be granted Logo

10GigE and above NICs that support all the required features but do not support all of the small cloud features for 10GigE NICs listed as being required for Cloud Ready, will be granted Logo

10GigE and above NICs that support all the required features as well as all of the features for small cloud NICs, will be granted Cloud Ready Additional Qualification and be supported for smaller cloud deployments

10GigE and above NICs that support all the required features as well as all of the features for large cloud NICs, will be granted Cloud Ready Additional Qualification and be supported for larger cloud deployments

Send comments about this topic to Microsoft

Device.Network.LAN.Base

LAN requirements

In this topic:

Device.Network.LAN.Base.100MbOrGreater

Ethernet devices must support 100-Mb or greater link speeds.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices must be able to link at 100 Mb or higher speeds.

Device.Network.LAN.Base.32MulticastAddresses

Ethernet devices must support filtering for at least 32 multicast addresses.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

An Ethernet device must support filtering of 32 or more multicast addresses. Design Notes: See the Windows Driver Kit, "multicast." See the Windows Driver Kit, "NdisReadNetworkAddress" and "MAC Address."

Device.Network.LAN.Base.AdvProperties

Ethernet devices must safeguard advanced properties and provide complete configurability through advanced properties.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices must adhere to the standardized registry keywords for controlling advanced features as documented on MSDN. Devices must also safeguard both Microsoft standardized and private keywords from malicious values.

Device.Network.LAN.Base.AnyBoundary

Ethernet devices must be able to transmit packets from buffers aligned on any boundary.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Buffer alignment refers to whether a buffer begins on an odd-byte, word, double-word, or other boundary. Devices must be able to transmit packets with any of the packets fragments beginning on an odd-byte boundary. For performance reasons, packets must be received into contiguous buffers on a double-word boundary.

Device.Network.LAN.Base.IPv4AndIPv6OffloadParity

Ethernet devices that implement offloads must do so consistently for both IPv4 and IPv6.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Network offloads implemented by Ethernet devices need to operate consistently, irrespective of the IP protocol used. Having offload parity allows Windows customers to have a consistent and predictable experience across both IPv4 and IPv6.

Device.Network.LAN.Base.NDISCalls

Ethernet devices must make only NDIS library or WDF system calls.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A driver for an Ethernet device must make only NDIS or WDF calls. Any calls to other kernel mode components are not allowed. Design Notes: See the Windows Driver Kit, "NDIS" and "WDF."

Device.Network.LAN.Base.NDISRequirements

Ethernet devices must conform to the NDIS requirements in the Windows Driver Kit.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All Ethernet device drivers must conform to NDIS specified in the Windows Driver Kit. Design Notes: See the Windows Driver Kit, "NDIS."

Device.Network.LAN.Base.PacketFiltering

Ethernet devices must support packet filtering.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The miniport driver must support all filter types in the Windows Driver Kit. Note: Filtering should be performed in Hardware/Firmware.

Device.Network.LAN.Base.PreserveOSServices

Ethernet devices Miniport Driver/Driver Software must not disable OS services.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices Miniport Driver/Driver Software must not disable OS services. Some devices tend to shutoff services such as the Base Filtering Engine (BFE). This leaves the system vulnerable to attack due to lack of security capabilities.

Device.Network.LAN.Base.PriorityVLAN

Ethernet devices that implement link speeds of gigabit or greater must implement Priority & VLAN tagging according to the IEEE 802.1q specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement link speeds of gigabit or greater. If the Ethernet device does not implement link speeds of gigabit or greater, then this requirement does not apply. The Ethernet device and driver must support the inserting and removing of priority and VLAN tags.

Device.Network.LAN.Base.ShortPacketPadding

Ethernet devices must pad short packets with constant data.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

Padding that is added to short Ethernet packets to bring that packet size to the minimum IEEE 802.3, Section 4.2.3.3, packet size must be initialized to either zeros or an 8-bit repeating pattern before the packets are transmitted. The 802.3 Ethernet specification requires packets that are shorter than the minimum packet size to be padded up to the minimum size before the packets are transmitted onto the wire. The 802.3 Ethernet specification does not specify the padding content; however, Microsoft requires the padding to be zeros or another constant value to address security concerns. Some drivers pad the packets with data from previously sent packets; this practice is not acceptable. Design Notes: New solutions are recommended to implement a padding of zeros. However, some devices that implement the padding in hardware use 0xffs, which addresses the security concern.

Device.Network.LAN.Base.SupportIEEEE8023

Ethernet devices must comply with IEEE 802.3.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All 802.3 Ethernet devices must implement and comply with the IEEE 802.3 specification.

Send comments about this topic to Microsoft

Device.Network.LAN.ChecksumOffload

Network requirements

In this topic:

Device.Network.LAN.ChecksumOffload.ChecksumOffload

Ethernet devices must implement Checksum Offloads.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Send and Receive TCP Checksum Offload for IPv4 and IPv6

Send and Receive IP Checksum Offload for IPv4

Send and Receive UDP Checksum offload for IPv4 and IPv6

Support for TCP Checksum Standardized Keywords is mandatory.

Ethernet devices implement Checksum Offloads must expose the NDIS Enumeration Keywords.

Send comments about this topic to Microsoft

Device.Network.LAN.CS

A connected standby capable computer (also known as a platform) supports a low power active state, and advertises support of that state to the OS using the appropriate ACPI flag (LOW_POWER_S0_IDLE_CAPABLE) in FADT. It is also expected to meet all the Windows certification requirements for a connected standby platform. This section specifies the connected standby requirements for Wired LAN (Ethernet).

In this topic:

Device.Network.LAN.CS.NetworkWake

Wired LAN (Ethernet) devices integrated into Connected Standby systems or docks shipped with the system must support network wake patterns.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The specific requirements are listed below:

Wake on LAN Patterns:

Wired LAN devices and their drivers are required support at least thirty-two (32) bitmap patterns of a minimum of 128 byte size. This pattern type refers to flexible bitmap patterns (NDIS_PM_WOL_PACKET.NdisPMWoLPacketBitmapPattern) and not other pattern types.

Wake on Magic Packet:

Wired LAN devices and their drivers must support magic packet wake. Support for this wake packet type is required and is not included in the 32-bitmap pattern requirement.

Wake Packet Indication:

Wired LAN devices and their drivers are required to support Wake Packet Indication for all network wake packets and be capable of caching and indicating the complete network packet causing the wake. Note that this supersedes the Device.Network.LAN.PM.WakePacket requirement for Connected Standby-capable devices.

Design Notes:

See the Power Management specification on MSDN.

Device.Network.LAN.CS.PresenceOffload

Wired LAN (Ethernet) devices integrated into Connected Standby systems or docks shipped with the system must support network presence offload.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Support of this feature is required. The specific requirements are listed below:

ARP Protocol Offload:

Wired LAN devices must implement ARP offload as it is defined in the Power Management specification on MSDN. Specifically, the offload must respond to an ARP Request (operation = 1) by responding with an ARP Reply (operation = 2) as defined in RFC 826 (http://tools.ietf.org/html/rfc826).

NS Protocol Offload:

Wired LAN devices must implement IPv6 NS offload as it is defined in Power Management specification on MSDN. Specifically, the offload must respond to a Neighbor Solicitation (operation = 135) by responding with an NS Advertisement (operation = 136) as defined in RFC 4861 (http://tools.ietf.org/html/rfc4861). We require support for at least 2 ND offload requests. Each request can have 2 target addresses. The value they should advertise in NDIS_PM_CAPABILITIES::NumNSOffloadIPv6Addresses is the NUMBER OF REQUESTS, not number of addresses. So, if they support the minimum 2 requests, they should advertise 2. The name of the field is wrong and will be fixed in the next NDIS release.

The miniport must implement the said protocol in accordance to RFCs describing Neighbor Discovery and Neighbor Solicitation Protocol for IPv6.

Device.Network.LAN.CS.ReliableCSConnectivity

LAN device on systems that support Connected Standby must deliver reliable connectivity in Connected Standby.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The wired LAN device seamlessly transitions between working power state D0 and low power state D2/D3 while in Connected Standby (CS). The wired LAN device maintains OSI layer 2 link connectivity while in CS. The wired LAN device wakes up on matching wake patterns only. There are no spurious wakes while in CS. Wake packets are delivered without delay or buffering. Lock screen apps stay connected with Control Channel or Push Notification triggers over IPv4 and IPv6 in CS.

The exact D-value in low power state depends on the underlying bus type. For example, network adapters on USB, or SDIO buses use D2 as their low power state, while network adapters on PCI or PCIe buses use the D3 state.

Additional Information

Exceptions - Does not apply to non-AOAC capable devices

Device.Network.LAN.CS.WakeEvents

Wired LAN (Ethernet) devices integrated into Connected Standby systems or docks shipped with the system must support various wake triggers.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The specific requirements are listed below: Wake on Media Connect:Wired Ethernet devices must advertise and support wake on media connect as defined in the specification on MSDN. Wake on Media Disconnect: Wired Ethernet devices must advertise and support wake on media disconnect as defined in the specification on MSDN.Design Notes: See the Power Management specification on MSDN.

Device.Network.LAN.CS.WakeReasonDetection

Wired LAN (Ethernet) devices integrated into Connected Standby systems or docks shipped with the system must support wake reason detection.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The specific requirements are listed below: Wake Reason support: Wired Ethernet devices must include Wake Reason support in compliance with the NDIS_STATUS_PM_WAKE_REASON documentation on MSDN. When the wake is caused by an incoming network packet, the NIC is required to capture and indicate the entire packet causing the wake to the operating system. Design Notes: See the Power Management specification on MSDN.

Send comments about this topic to Microsoft

Device.Network.LAN.DCB

LAN requirements

In this topic:

Device.Network.LAN.DCB.DCB

Ethernet devices that implement Data Center Bridging (DCB) must comply with the DCB Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that support Data Center Bridging (DCB).

Ethernet devices that implement Data Center Bridging (DCB) must comply with the following requirements:

MaxNumTrafficClasses must be between 3 and 8 inclusive.

MaxNumEtsCapableTrafficClasses must be at least 2.

MaxNumPfcEnabeldTrafficClasses must be at least 1.

ETS Must be supported.

PFC Must be supported.

Strict Priority Must be supported.

iSCSI CNAs must support classification element for iSCSI traffic (TCP ports 860 and 3260, both src and dest port).

FCoE CNAs must support classification element for FCoE traffic (Ethertype of 0x8906).

Design Notes

See the Data Center Bridging Specification at http://msdn.microsoft.com/en-us/library/windows/hardware/hh451635(v=vs.85).aspx.

Send comments about this topic to Microsoft

Device.Network.LAN.GRE

LAN requirements

In this topic:

Device.Network.LAN.GRE.GREPacketTaskOffloads

Ethernet devices that implement GRE Encapsulated Packet Task Offloads must comply with the specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement GRE encapsulated packet task offloads. Ethernet devices that implement GRE encapsulated packet task offloads must comply with the specification and support the following encapsulated offload tasks for all combinations of inner/outer IPv4/IPv6 headers:

Send checksum (IPv4, TCP, UDP)

Receive checksum (IPv4, TCP, UDP)

LSOv2

VMQ

Send comments about this topic to Microsoft

Device.Network.LAN.IPsec

LAN requirements

In this topic:

Device.Network.LAN.IPsec.IPsec

Ethernet devices that implement IPsec task offload must support required modes and protocols.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices that support IPsec task offload must support the following: Version: IPsec Task offload v2 NDIS version: 6.1, 6.20, 6.30.

Ethernet devices that support IPsec task offload for Windows 8 must use NDIS 6.30. Mode:

- IPv4 and IPv6 Transport

- IPv4 and IPv6 TunnelProtocol: ESPCrypto Algorithm:

- Must implement: AES-GCM (128 or higher)

- May implement additional algorithms as stated in http://technet.microsoft.com/en-us/library/dd125380(v=WS.10).aspxCoexistence: Ethernet devices that support IPsec task offload and any the following offload technologies:

- Large Send Offload- Receive Side Scaling

- CheckSum offload.Must implement them in a coexisting manner, such that the use of IPsec task offload does not preclude the use of the other offload technologies implemented for each IPsec mode.

Send comments about this topic to Microsoft

Device.Network.LAN.KRDMA

LAN requirements

In this topic:

Device.Network.LAN.KRDMA.KRDMA

Devices that implement the NetworkDirect Kernel Mode Interface (NDKPI) (a.k.a., Kernel-mode RDMA, kRDMA) must comply with the Network Direct Kernel Mode Interface (NDKPI) Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement the Network Direct Kernel Mode Interface (NDKPI) Specification. Devices that implement Network Direct Kernel Mode Interface (NDKPI) (a.k.a., Kernel-mode RDMA) must comply with the Network Direct Kernel Mode Interface (NDKPI) Specification, version 1.2.

Design Notes:

See the Network Direct Kernel Mode Interface (NDKPI) Specification.

Send comments about this topic to Microsoft

Device.Network.LAN.LargeSendOffload

Network requirements

In this topic:

Device.Network.LAN.LargeSendOffload.LargeSendOffload

Ethernet devices must implement large send offloads.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices must support the following task offloads:

Large Send Offload version 2 for IPv4 and IPv6.

Design Notes:

See the Windows Driver Kit, "NDIS."

Send comments about this topic to Microsoft

Device.Network.LAN.MTUSize

Device.Network.LAN.MTUSize

Applies to	Windows Server 2016 Technical Preview x64

Description

Ethernet devices must support Jumbo Frames. MTU values in the User Interface must include the Ethernet header size of 14 Bytes. The “*JumboPacket” standardized keyword in the Windows Registry is currently used for setting the MTU size and should remain as an enumerable value with supported values being 1514, 4088, and 9014. A new keyword will be added called “*EncapOverhead” with type being numeric, default value of 0, max value of 480, and step increments of size 32. This value will account for overhead in Ethernet frames due to virtual network overlay encapsulation such as VXLAN and NVGRE. MTU size will therefore be determined by the summation of these two keyword values except perhaps in the case where *JumboPacket + *EncapOverhead would exceed some hardware upper bound of the NIC.

Design Notes:

TBA

Send comments about this topic to Microsoft

Device.Network.LAN.PM

LAN requirements

In this topic:

Device.Network.LAN.PM.PowMgmtNDIS

Ethernet devices that implement network presence offloads must conform to the Power Management specification on the NDIS program.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices must implement ARP offload as defined in the Power Management specification on MSDN. Specifically, the offload must respond to an ARP Request (operation = 1) by responding with an ARP Reply (operation = 2) as defined in RFC 826.

Ethernet devices must implement IPv6 NS offload as defined in the Power Management specification on MSDN. Specifically, the offload must respond to a Neighbor Solicitation (operation = 135) by responding with an NS Advertisement (operation = 136) as defined in RFC 4861. Devices must support at least 2 NS offloads, each with up to 2 target IPv6 addresses.

Design Notes

See the Power Management specification on MSDN.

See RFC 826 at http://go.microsoft.com/fwlink/?LinkId=108718.

Exceptions - Exceptions to this requirement include: PC Card, CardBus devices and for external USB network devices operating on bus power. Devices with transmission speeds in excess of 1 gigabit. Devices with fiber optic medium.

Device.Network.LAN.PM.WakeOnLANPatterns

Ethernet devices must implement Wake on LAN patterns according to the specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Implementation must comply with Network Device Class Power Management Reference Specification. Ethernet devices and drivers are required to support at least eight (8) bitmap patterns since Windows uses up to eight patterns. Magic packet wake support is required and is not included in the 8-bitmap pattern requirement.

Design Notes:

Implementation details are in the Power Management specification on MSDN.

Exception - Exceptions to this requirement include: PC Card, CardBus devices and for external USB network devices operating on bus power. Devices with transmission speeds in excess of 1 gigabit. Devices with fiber optic medium. Note: If the device implements Wake on LAN - it must implement it correctly based on this requirement regardless of whether the device is on the exception list.

Device.Network.LAN.PM.WakePacket

Ethernet devices that implement Wake Packet Detection must comply with Network Device Class Power Management Reference Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices that implement Wake Packet Detection must comply with Network Device Class Power Management Reference Specification. Implementation must comply with Network Device Class Power Management Reference Specification discussed in the Windows WDK. The NIC is required to capture at least the first 128 bytes of the packet causing the network wake and generate a status indication to the operating system.

Design Notes:

See the WDK, Network Device Class Power Management Reference Specification.

Send comments about this topic to Microsoft

Device.Network.LAN.RSC

LAN requirements

In this topic:

Device.Network.LAN.RSC.RSC

Ethernet devices that implement Receive Segment Coalescing (RSC) must comply with the RSC Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices that implement Receive Segment Coalescing (RSC) must comply with the RSC Specification and require both IPv4 and IPv6 support.

Design Notes:

NDIS version: 6.30

Send comments about this topic to Microsoft

Device.Network.LAN.RSS

LAN requirements

In this topic:

Device.Network.LAN.RSS.RSS

Ethernet devices that implement RSS

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

RSS support is required on server SKUs for all devices except for SR-IOV VF drivers.

RSS support is optional on client SKUs.

Ethernet devices that implement RSS must support:

Hash types: IPv4, TCP IPv4, IPv6, and TCP IPv6

Multiple processor groups (for miniports of NDIS version 6.20 and above)

Number of queues (depending on link speed):

Less than 10 gigabit: 2

10 gigabit or greater: 8

SR-IOV VF driver (independent of link speed): 2

Indirection table size (depending on link speed):

Less than 10 gigabit: 64

10 gigabit or greater: 128

SR-IOV VF driver (independent of link speed): 16

SR-IOV PF driver (independent of link speed): 64

Implement all RSS standardized keywords

*RSS

*NumRSSQueues

Message Signaled Interrupts Extended (MSI-X) as defined in the PCI v3.0 specification. For devices with a link speed of less than 10 gigabit, the device must have 1 MSI-X vector with support for 2 RSS hardware queues. For devices 10 gigabit or greater, the device must have an MSI-X vector for each RSS hardware queue. For example, if the device has a link speed of 10 gigabits and advertises support for 8 RSS hardware queues then it must have at least 8 MSI-X vectors in the hardware.

Design Notes:

See RSS Standardized Keywords Specification
http://msdn.microsoft.com/library/windows/hardware/ff570864(v=vs.85).aspx.

In addition, the device must allocate as many MSI-X table entries as there are CPUs in the system. See the NDIS documentation section on MSI-X for more details
http://msdn.microsoft.com/library/windows/hardware/ff566491.aspx.

Device.Network.LAN.RSS.SetHashFunctionTypeAndValue

Ethernet devices that implement RSS must set the hash function, hash type, and hash value on each indicated packet.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement RSS. If the Ethernet device does not implement RSS, then this requirement does not apply. If the network device supports RSS, all packets for which the RSS implementation was able to calculate the hash, the RSS implementation must return the full 32-bit hash value, the hash function, and the hash type, for each received packet it indicates. For any packets it received without error for which it was not able to generate the hash function, it must clear the hash type. Macros NET_BUFFER_LIST_SET_HASH_TYPE,NET_BUFFER_LIST_SET_HASH_FUNCTION, and NET_BUFFER_LIST_SET_HASH_VALUE must be used to set the associated values.

Design Notes:

See the MSDN page for more information: http://msdn.microsoft.com/en-us/library/windows/hardware/ff570726(v=vs.85).aspx. See the Windows Driver Kit, "Indicating RSS Receive Data."

Device.Network.LAN.RSS.SupportIndirectionTablesSizes

Ethernet devices that implement RSS must support specific Indirection Table sizes.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement RSS. If the Ethernet device does not implement RSS, then this requirement does not apply. If the network device supports RSS, the RSS implementation must support indirection table sizes for each power of 2, up to the maximum indirection table size supported. For example, if 128 is the maximum indirection table size, then the miniport must accept indirection tables of sizes 2, 4, 8, 16, 32, 64, or 128. The lookup into the Indirection Table to find the destination CPU must be accomplished by using only the least significant bits as specified by the last value set in the OID_GEN_RECEIVE_SCALE_PARAMETERS, NumberOfLsbs variable. An RSS implementation must support the host protocol stack setting NumberOfLsbs to any value between 1 and 7, inclusive.

Design Notes:

See the Windows Driver Kit, "OID_GEN_RECEIVE_SCALE_PARAMETERS."

Device.Network.LAN.RSS.SupportToeplitzHashFunction

Ethernet devices that implement RSS must support the Toeplitz hash function.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement RSS. If the Ethernet device does not implement RSS, then this requirement does not apply. If the network device supports RSS, the RSS implementation must at least support the Toeplitz hash function for the types of packets for which it advertised as being able to generate the hash (as specified in OID_GEN_RECEIVE_SCALE_CAPABILITIES). This includes support for the HashSecretKey length of 40 bytes.

Design Notes:

See Windows Driver Kit, "RSS Hashing Functions." Also, refer to MSDN for more information http://msdn.microsoft.com/en-us/library/windows/hardware/ff570725(v=vs.85).aspx

Device.Network.LAN.RSS.SupportUpdatesToRSSInfo

Ethernet devices that implement RSS must support updates to RSS information at any time.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement RSS. If the Ethernet device does not implement RSS, then this requirement does not apply. At any time a network device supports RSS, it must support setting OID_GEN_RECEIVE_SCALE_PARAMETERS, including updating the Indirection Table, NumberOfLsbs, SecretKey, and HashInformation (hash function and hash type). The RSS implementation can post packets out of order during the transition from the prior state to the new state and can perform a hardware reset if the HashInformation, SecretKey, or NumberOfLsbs changed. It must not perform a hardware reset if only the Indirection Table contents are changed.

Design Notes:

See the Windows Driver Kit, "OID_GEN_RECEIVE_SCALE_PARAMETERS."

Send comments about this topic to Microsoft

Device.Network.LAN.SRIOV

Network requirements

In this topic:

Device.Network.LAN.SRIOV.SRIOV

Ethernet devices that implement Single Root I/O Virtualization (SR-IOV) must support required functionalities.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A driver must use Microsoft software back-channel to PF for VF driver PCIe configuration space requests.

The default initial switch configuration must be in the INF file.

The PF INF must set the UpperRange to "ndis5" and LowerRange to "ethernet".

The INF must not include the *SriovPreferred keyword.

Coalesced filters must be set in NDIS_REECIVED_FILTER_CAPABILITIES.

SR-IOV NIC must include a Virtual Ethernet Bridge (VEB).

If RSS is supported for VF miniports, the NIC must be able to support an independent RSS hash for each function, physical or virtual.

Both the PF and VF miniport drivers must be able to pass the LAN logo tests.

The default vPort cannot be deleted; non-default vPorts on a VF can be deleted.

If SRIOV is disabled, the NIC and miniport must function as a standard Ethernet NIC.

An SRIOV NIC must also advertise and implement VMQ. Queue pair not allocated to IOV are to be available for VMQ.

On report of media connected, the VF miniport must be ready to accept traffic.

A VF miniport must specify an UpperRange of "ndisvf" and a LowerRange of "iovvf".

Design Notes:

See the Single Root I/O Virtualization Specification.

Send comments about this topic to Microsoft

Device.Network.LAN.SRIOV.VF

Network requirements

In this topic:

Device.Network.LAN.SRIOV.VF.VF

Ethernet devices that implement Single Root I/O Virtualization (SR-IOV) must support required functionalities.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Requirements for an SRIOV VF adapter

A driver must use Microsoft software back-channel to PF for VF driver PCIe configuration space requests.

If RSS is supported for VF miniports, the NIC must be able to support the same number of RSS as it can on PFs.

All NDIS device must conform to NDIS 6.x specified in the Windows Driver Kit so should the VF devices.

A VF miniport must specify an UpperRange of "ndisvf" and a LowerRange of "iovvf".

If the VF miniport implements Receive Segment Coalescing (RSC), it must comply with the RSC specification and require both IPv4 and IPv6 support.

On report of media connected, the VF miniport must be ready to accept traffic.

Design Notes:

See the Single Root I/O Virtualization Specification.

Send comments about this topic to Microsoft

Device.Network.LAN.TCPChimney

TCP Chimney requirements

In this topic:

Device.Network.LAN.TCPChimney.ComplyWithNDIS

Ethernet devices that implement TCP Chimney must comply with the latest NDIS miniport driver model.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement TCP Chimney. If the Ethernet device does not implement TCP Chimney, then this requirement does not apply. The TCP Chimney portion of the device must comply with the TCP Chimney specification on Connect.

Design Notes:

See Windows Driver Kit, "Network Devices and Protocols."

Device.Network.LAN.TCPChimney.ComplyWithTCPIPProtocol

Ethernet devices that implement TCP Chimney must comply with the IETF standard RFCs for the TCP/IP protocol family and behave as the Microsoft Windows (host) TCP/IP protocol implementation.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement TCP Chimney. If the Ethernet device does not implement TCP Chimney, then this requirement does not apply. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT","SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this requirement are to be interpreted as described in RFC 2119. A TCP chimney NIC MUST implement the TCP/IP protocol family such that: 1. The TCP/IP protocol implementation conforms to the IETF standard RFCs and 2. The TCP/IP protocol implementation behaves in the same way as the Microsoft Windows TCP/IP protocol implementation. This requirement specifies which RFCs must be implemented by the TCP chimney NIC and clarifies the expected behavior in cases where an RFC is ambiguous. Table 1 lists the RFCs that a TCP Chimney NIC must implement.

Descriptive name	Specification
Transmission Control Protocol RFCs	RFC 793 - Transmission Control Protocol RFC 813 - Window and Acknowledgment Strategy in TCP RFC 1122* - Requirements for Internet Hosts - Communication Layers - entire section 4.2

	RFC 1323 - TCP Extensions for High Performance RFC 2923* -TCP Problems with Path MTU Discovery RFC 2988* - Computing TCP's RTO RFC 3465* - Appropriate Byte Counting
TCP congestion control	RFC 2581* - TCP Congestion Control RFC 2582* - New Reno Modification to TCP's Fast Recovery Alg. RFC 3042 - Limited Transmit
TCP loss recovery	RFC 2018* - TCP Selective Acknowledgment Options RFC 3517* - Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP
TCP security	RFC tcpm-tcpsecure-09* - Improving TCP's Robustness to Blind In-Window Attacks
Internet Protocol v4 RFCs**	RFC 791, RFC 894, RFC 1042, RFC 1191, RFC 1122 - entire section 3.2
Internet Protocol v6 RFCs**	RFC 1752, RFC 1981, RFC 2374, RFC 2460, RFC 2461, RFC 2675, RFC 2711, RFC 3122, RFC 3513
* - There are associated clarifications for the RFC, which must be followed. They are outlined below.** - TCP Chimney NICs MUST NOT implement the entire set of IP related RFCs. Instead, the TCP Chimney Driver Kit guidelines for the Internet Protocol RFC implementation must be followed.	

Table 1 - Lists of RFCs that a TCP Chimney NIC must implement RFC Clarifications. The following clarifications must be followed by the TCP/IP implementation in the TCP Chimney NIC. RFC 11221. Section 4.2.3.4 specifies that the Nagle algorithm SHOULD be implemented as a method to avoid the Silly Window Syndrome. The TCP chimney NIC MUST implement the Nagle algorithm and the

implementation must follow this pseudo code: a. When sending a segment, the first stage of SWS avoidance MUST be implemented as:

Send(){..If (BytesToSend > MSS ||BytesToSend > MaxSndWnd /2 ||BytesToSend >= BytesInCurReq ||ForceOutput){BeginSend();}else{StartSwsTimer();} ...

...

BytesToSend The number of available bytes that can be sent as allowed by the current send window.

MSS Maximum Segment

SizeMaxSndWnd The maximum receive window that the TCP peer ever advertised.

BytesInCurReq Bytes left in the current send request.

ForceOutput A variable that determines if the segment MUST be sent, due to SWS timer expiring as an example. The line in red specifies the deviation from the SWS avoidance that MUST be implemented.

Note: This pseudo code defines the behavior at the time of sending, not at the time when the send request is offloaded by the host TCP/IP stack. The reason why the Microsoft TCPIP stack deviates from the SWS algorithm in the way described above is:

1. CWND can grow in Bytes. More precisely, CWND is not constrained to grow or shrink in multiples of MSS or PUSH boundaries. Because the TCP implementation in Windows implements Appropriate Byte Counting (RFC 3465), this point is strengthened even further.

2. The PUSH boundary is determined by the TCP application posting data to be sent so it is not guaranteed to be aligned with the MSS size.

3. Because of #1 and #2 it is very likely for the TCP state machine to reach a point at which one MSS has been placed on the wire and there is a sub-MSS segment which, if sent, will complete the block of data up to the PUSH boundary.

    a. In this case, it is favorable for TCP to send this one sub-MSS segment in order to complete the transmission of the app's buffer up to the PUSH boundary. The reason why it is favorable to do this is because the data will be delivered to the receiving application faster than if the SWS algorithm was followed to the letter. At the same time, the deviation does not re-introduce any of the problems SWS addressed in the first place.

4. As described in section 4.2.2.17 a TCP Chimney NIC MUST use the connection RTT as a trigger to send a zero window probe and then exponentially increase the interval between successive probes. In addition, the probe MUST contain 1 new Byte of data.

5. TCP Chimney NIC MUST support filling at least two reassembly holes. RFC 20181. The TCP Chimney NIC MAY implement RFC 2018. If a TCP Chimney NIC implements RFC 2018, then it MUST also implement RFC 3517.2. A TCP Chimney NIC that DOES NOT implement RFC 2018 MUST properly process pure ACK packets, which contain SACK blocks, as described in section 3 of RFC 793.RFC 25811. The TCP Chimney NIC MUST be able to transition from using the slow-start algorithm to using the congestion avoidance algorithm as specified in Section 3.1. In addition it MUST implement Congestion Window (cwnd) = Slow Start Threshold (ssthresh) instead of Congestion Window > Slow Start Threshold.RFC 25821. The TCP Chimney NIC MUST use the following equation instead of the one described in RFC 2582, section 3 - point 1:SsThresh = max(2*mss, min(cwnd,window_advertised_by_peer)/2)RFC 29231. The TCP Chimney NIC is NOT required to

implement the recommendations outlined in RFC 2923. Instead, the TCP Chimney NIC must upload the TCP connection to allow the host stack to execute the black hole detection state machine. See the Windows Driver Kit for details.RFC 29881. See RFC 1122 section 4.2.3.1 and RFC 2988 for background information. TCP Chimney NIC MUST implement RTO calculation using the following algorithm, which is the same as RFC 2988 with minor exceptions that are qualified below:function CalculateRto (first, byRef srtt, byRef rttvar, m)rttSample = Minimum (m, 30s)if first thenrttvar = m/2srtt = melse' notice that rttvar is calculated first, using the old ' value of srttrttvar = (3/4) * rttvar + (1/4) * abs(srtt - rttSample)srtt = (7/8)*srtt + (1/8) * rttSampleend ifCalculateRto = srtt + 4 * rttvarCalculateRto = Minimum (CalculateRto, 60s)CalculateRto = Maximum (CalculateRto, 300ms)The two lines in red capture the deviation from the RFC. Specifically, it is expected that the TCP Chimney NIC has an upper bound when calculating the RTT value.RFC 34651. Section 2.1 describes the changes to CWND during congestion avoidance. A TCP Chimney NIC MUST use the following formula to calculate CWND during congestion avoidance:// L is 4CWnd += max((MaxMss * min(MaxMss * L, BytesAcked)) /CWnd, 1)Note that if BytesAcked is always 1 the above equation becomes max((MaxMss, MaxMss)/Cwnd, 1)which is equivalent to equation 2 in RFC 2581. 2. Section 2.3 in RFC 3465 discusses the limit, L, chosen for the CWND increase during slow start and congestion avoidance, which controls the aggressiveness of the algorithm. A TCP Chimney NIC MUST use a value of 4 for L in order for it to exhibit the same behavior as the Windows TCP/IP protocol implementation.RFC tcpm-tcpsecure-091. TCP Chimney NICs MUST follow the security guidelines outlined in sections 3, 4, and 5 of the 'TCP Security' internet draft RFC (http://tools.ietf.org/html/draft-ietf-tcpm-tcpsecure-12). 2. TCP Chimney NICSs SHOULD follow the Windows specific implementation details described in the WDK.Design Notes: See the full text of the RFCs at http://go.microsoft.com/fwlink/?LinkId=36702.

Device.Network.LAN.TCPChimney.HandlesOutOfOrderData

Ethernet devices that implement TCP Chimney must properly handle the Out Of Order data scenarios.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Ethernet devices that implement TCP Chimney must properly handle Out Of Order data scenarios described below:

1. If anything is placed in the reassembly queue after an inorder FIN, then the reassembly queue MUST be flushed by discarding all of its contents.

2. If a TCP Chimney NIC stores an OOO FIN in the reassembly queue, then it MUST not store OOO data or OOO FIN beyond another OOO FIN in the reassembly queue. If it receives OOO data or OOO FIN segment that would lead to such a conflict, then the TCP Chimney NIC MUST drop that segment and flush the reassembly queue by discarding all of its contents.

Device.Network.LAN.TCPChimney.ImplementSufficientlyGranularTimers

Ethernet devices that implement TCP Chimney must implement sufficiently granular timers.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement TCP Chimney. If the Ethernet device does not implement TCP Chimney, then this requirement does not apply. The TCP chimney NIC must have access to timers (implemented on the NIC's hardware) with precise enough granularity and skew such that it can drive the TCP/IP state machine correctly. The timer granularity must be 10 milliseconds or better (lower than 10 ms) and the timer skew must be as good as what general purpose CPU timer provides.

Device.Network.LAN.TCPChimney.NeighborStateObjTimestampsComplyWithWDK

Neighbor state object timestamps are implemented according to details in the Windows Driver Kit.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A network device that implements TCP Chimney must ensure that TCP Chimney maintains a timestamp for each neighbor state object and perform checks against the timestamp on each incoming and outgoing packet.

Design Notes:

See the Windows Driver Kit, "OID_TCP_OFFLOAD.

Device.Network.LAN.TCPChimney.Support1024Connections

Ethernet devices that implement TCP Chimney must support at least 1024 connections and not advertise more offload capacity than what the hardware can support.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement TCP Chimney. If the Ethernet device does not implement TCP Chimney, then this requirement does not apply. Ethernet devices

that implement TCP Chimney must support at least 1024 connections and not advertise more offload capacity than what the hardware can support.

Device.Network.LAN.TCPChimney.Support64bitAddresses

Ethernet devices that implement TCP Chimney must support 64-bit addresses.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement TCP Chimney. If the Ethernet device does not implement TCP Chimney, then this requirement does not apply. If the device uses PCI, it must support 64-bit addresses; 64-bit data support is not required.

Send comments about this topic to Microsoft

Device.Network.LAN.VMQ

LAN requirements

In this topic:

Device.Network.LAN.VMQ.VirtualMachineQueues

Ethernet devices that implement Virtual Machine Queues must comply with the Programmable Machine Queues Reference specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The implementation must comply with the Programmable Machine Queues Reference Specification.

At least four queues with filters must be supported. Support for at least 16 queues with filters is recommended. The number of queues required will be 16 by December 1, 2009 for 10 Gigabit parts. The number of queues required is inclusive of the default queue.

MSI-X Support (NDIS_RECEIVE_FILTER_MSI_X_SUPPORTED) is mandatory:

Queues	MSI-X to Queues Ratio
1 to 16	1:1
17-64	1:2 (Min 16)
65-unlimited	1:16 (Min 32)

Filtering:

Support for VLAN filtering in HW (NDIS_RECEIVE_FILTER_MAC_HEADER_VLAN_ID_SUPPORTED) is optional. If implemented, VLANs per VM Queue (NumVlansPerVMQueue) must be >= 1

Support for MAC filtering in HW (NDIS_RECEIVE_FILTER_MAC_HEADER_DEST_ADDR_SUPPORTED) is mandatory.

Support for NDIS_RECEIVE_FILTER_TEST_HEADER_EQUAL_SUPPORTED is mandatory.

The maximum number of MAC header filters (MaxMacHeaderFilters) must be >= Number of queues

Total MAC addresses (NumTotalMacAddresses) must be >= Number of queues

MAC addresses per VM Queue (NumMacAddressesPerVMQueue) must be >= 1

Per-queue receive indication must be supported (NDIS_RECEIVE_QUEUE_PARAMETERS_PER_QUEUE_RECEIVE_INDICATION)

Dynamic VMQ support is required for Windows Server 2012 only.

Design Notes

Implementation details are in the ProgrammableMachine Queues specification, on the NDIS Program, Connect site http://msdn.microsoft.com/en-us/library/windows/hardware/ff571034(v=vs.85).aspx

Send comments about this topic to Microsoft

Device.Network.LAN.VXLAN

In this topic:

Device.Network.LAN.VXLAN.VXLANPacketTaskOffloads

Applies to	Windows Server 2016 Technical Preview x64

Description

This requirement only applies to Ethernet devices that implement VXLAN encapsulated packet task offloads. Ethernet devices that implement VXLAN encapsulated packet task offloads must comply with the specification and support the following encapsulated offload tasks for all combinations of inner/outer IPv4/IPv6 headers:

Send checksum (IPv4, TCP, UDP)

Receive checksum (IPv4, TCP, UDP)

LSOv2

VMQ

Send comments about this topic to Microsoft

Device.Network.MobileBroadband.CDMA

Mobile broadband

In this topic:

Device.Network.MobileBroadband.CDMA.ComplyWithBaseReq

Mobile broadband devices must comply with the following base requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile Broadband devices must comply with the following base requirements:

- MUST conform to NDIS 6.30 and Microsoft Mobile Broadband Driver Model Specification requirements in the Windows Driver Kit.
- MUST comply with power management specifications.
- MUST meet the performance target for various operation specified for various class of devices in the Mobile Broadband Driver Model Specification.
- A mobile broadband device driver must implement and conform to the NDIS 6.30 and Microsoft Mobile Broadband Driver Model Specifications. All recommended implementation specified in the Mobile Broadband Driver Model Specifications must be implemented. Note that Microsoft's MB class driver is compliant to the preceding requirements. A mobile broadband device must support the Power Management Policy as outlined in the Network Device Class Power Management Reference Specification, Version 2.0. A device must be functional after various OS Power Management operations. A device must meet the performance targets described in the Mobile Broadband Driver Model Specification.

Design Notes:

Helpful links: Mobile Broadband Driver Model Specifications http://msdn.microsoft.com/en-us/library/ff560543(v=VS.85).aspx Network Device Class Power Management Reference Specification http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/netpmspc.rtf

Device.Network.MobileBroadband.CDMA.FWComplyWithMBSpec

USB interface based CDMA class of Mobile Broadband device firmware must comply with USB-IF's Mobile Broadband Interface Model Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The USB interface based CDMA class of Mobile Broadband device firmware implementation must comply with the USB-IF's Mobile Broadband Interface Model (MBIM) Specification. No additional IHV drivers are needed for the functionality of the device and the device must work with Microsoft's Mobile Broadband (MB) class driver implementation.

Device firmware must also comply with the MBIM Errata* in addition to the MBIM 1.0 specification. In specific, the following items in MBIM Errata need to be compliant with:

•MEFD (MB Extended Functional Descriptor): Devices with Operator specific firmware must report the correct MTU size as required by the Mobile Network Operator for carrier certified devices.

*USB-IF Link that is accessible only to NCM DWG members. This errata will be published once it gets approved.

Additional Details:

Mobile Broadband Interface Model Specification:
http://www.usb.org/developers/devclass_docs/MBIM10.zip

Mobile Broadband Driver Model Specification:
http://msdn.microsoft.com/library/windows/hardware/ff560543(v=VS.85).aspx

Device.Network.MobileBroadband.CDMA.IdentityMorphing

Mobile Broadband Devices must support Identity Morphing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile Broadband devices based on USB protocol must support Identity Morphing. Implementing this requirement in the device firmware enables the device manufacturers to take advantage of Microsoft's inbox MB Class Driver in Windows 8 and the flexibility of using their own driver for previous generations of Windows operating systems version 7 and below. Links to the relevant specifications are provided in the Additional Information section below. Additional Information: Identity Morphing Specification: See MSDN.

Device.Network.MobileBroadband.CDMA.Loopback

Mobile Broadband Devices based on USB protocol must implement loopback functionality for performance and payload conformance testing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile Broadband devices based on USB protocol must implement a loopback functionality as detailed in the specification in the device firmware. Note that Loopback functionality is only tested for the data packet as it is the one in the performance critical path. Devices must pass the loopback test for performance requirements.

Specifically, devices must be able to support combined throughput of 100 Mbps (50 Mbps uplink / 50 Mbps downlink) or above and up to 5% loss rate. Links to the relevant specifications are provided in the Additional Information section below.

Additional Information:

Loopback implementation guide for device firmware: http://msdn.microsoft.com/en-us/library/windows/hardware/hh975390.aspx

MB Miniport Driver Performance Requirements: http://msdn.microsoft.com/en-us/library/windows/hardware/ff557193(v=vs.85).aspx

Device.Network.MobileBroadband.CDMA.MultiCarrierFunctionality

Mobile broadband devices that support multi-carrier feature must support the multi-carrier functionality.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile Broadband devices that support multi-carrier feature must support the multi-carrier functionality and should also do the following:

- Must meet the multi-carrier performance requirements available in the Mobile Broadband Driver Model Specification.
- Must stay on the bus when changing home providers.
- Must successfully pass all applicable logo tests covering all the different cellular class technologies that the device is capable of connecting to.
- Mobile broadband devices supporting multi-carrier feature must meet the multi-carrier performance requirements specified in the mobile broadband driver model specification. Mobile broadband devices that support multi-carrier feature must not do a bus / device re-enumeration or power reset the device resulting in PnP re-enumeration to the Windows when changing the home providers. If the device is capable of supporting GSM and CDMA cellular class technologies, then the device must execute both GSM as well as CDMA logo tests. For this to be covered correctly, the location of logo test execution must be in the coverage area of at least one GSM and one CDMA cellular class technologies.

Device.Network.MobileBroadband.CDMA.ReliableCSConnectivity

Wireless WAN device on systems that support Connected Standby must deliver reliable connectivity in Connected Standby.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The device seamlessly transitions between D0 and D3 warm states while in Connected Standby (CS). The device maintains both L2 & L3 connectivity while in CS. The device wakes up on matching wake patterns only. There are no spurious wakes while in CS. The wake packets are delivered without delay or buffering. RealTimeCommunication apps stay connected in CS over IPv4 and IPv6.

Device.Network.MobileBroadband.CDMA.SupportUSBSelectiveSuspend

USB based mobile broadband devices must support Windows implementation of USB selective suspend.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB based mobile broadband devices must support Windows implementation of USB selective suspend (SS). No alternate USB SS implementation is allowed.

Device.Network.MobileBroadband.CDMA.SupportWakeOnMB

Mobile broadband class of devices must support the following wake on mobile broadband capabilities.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile broadband class of devices MUST support the following wake on mobile broadband capabilities.

- Devices MUST support 32 bitmap wake patterns of 128 byte each.

- Devices MUST wake the system on register state change.
- Devices MUST wake the system on media connect.
- Devices MUST wake the system on media disconnect.
- GSM and CDMA class of Devices MUST wake the system on receiving an incoming SMS message.
- Devices that support USSD MUST wake the system on receiving a USSD message.
- Devices MUST support wake packet indication. NIC should cache the packet causing the wake on hardware and pass it up when the OS is ready for receives.

Mobile class of devices must support wake on mobile broadband. A device should wake the system on above mentioned events. Note that wake on USSD is mandatory only if the device reports that it supports USSD; otherwise, it is optional. See the following MSDN documentation for more information on the SMS and register state wake events.

1.NDIS_STATUS_WWAN_REGISTER_STATE

2.NDIS_STATUS_WWAN_SMS_RECEIVE

Send comments about this topic to Microsoft

Device.Network.MobileBroadband.FirmwareUpdater

Mobile broadband

In this topic:

Device.Network.MobileBroadband.FirmwareUpdater.FirmwareUpgrade

USB interface based GSM and CDMA class of mobile broadband devices that comply with Microsoft's firmware update platform must implement Firmware ID Device Service and an UMDF based firmware update driver for the firmware payload update to the device.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB interface based GSM and CDMA class of mobile broadband devices that comply with Microsoft's firmware update platform must implement Firmware ID Device Service (to be published soon on MSDN) and an UMDF based firmware update driver (guidelines and sample to be published soon on MSDN) for the firmware payload update to the device.

Additional Information

Send comments about this topic to Microsoft

Device.Network.MobileBroadband.GSM

Mobile broadband

In this topic:

Device.Network.MobileBroadband.GSM.ComplyWithBaseReq

Mobile broadband devices must comply with the following base requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile Broadband devices must comply with the following base requirements:

- MUST conform to NDIS 6.30 and Microsoft Mobile Broadband Driver Model Specification requirements in the Windows Driver Kit.
- MUST comply with power management specifications.
- MUST meet the performance target for various operation specified for various class of devices in the Mobile Broadband Driver Model Specification.
- Mobile broadband device driver must implement and conform to the NDIS 6.30 and Microsoft Mobile Broadband Driver Model Specifications. All recommended implementation specified in the Mobile Broadband Driver Model Specifications needs to be implemented. Note that Microsoft's MB class driver is compliant to above requirements. A mobile broadband device must support the Power Management Policy as outlined in the Network Device Class Power Management Reference Specification, Version 2.0. Device must be functional after various OS Power Management operations. Device must meet the performance targets described in the Mobile Broadband Driver Model Specification.

Design Notes:

Helpful links: Mobile Broadband Driver Model Specifications http://msdn.microsoft.com/en-us/library/ff560543(v=VS.85).aspx Network Device Class Power Management Reference Specification http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/netpmspc.rtf

Device.Network.MobileBroadband.GSM.EAPSIM

GSM class of Mobile Broadband devices that support the extensible authentication protocol method for GSM Subscriber Identity Module (EAP-SIM) must support EAP-SIM defined in RFC 4186.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

GSM devices that support EAP-SIM must support EAP-SIM as defined in RFC 4186.

Device.Network.MobileBroadband.GSM.FWComplyWithMBSpec

USB interface based GSM class of Mobile Broadband device firmware must comply with USB-IF's Mobile Broadband Interface Model Specification.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

USB interface based GSM class of Mobile Broadband device firmware implementation must comply with the USB-IF's Mobile Broadband Interface Model (MBIM) Specification. No additional IHV drivers are needed for the functionality of the device and the device must work with Microsoft's Mobile Broadband (MB) class driver implementation.

Device firmware must also comply with the MBIM Errata* in addition to the MBIM 1.0 specification. In specific, the following items in MBIM Errata need to be compliant with:

- MEFD (MB Extended Functional Descriptor): Devices with Operator specific firmware must report the correct MTU size as required by the Mobile Network Operator for carrier certified devices.

*USB-IF Link that is accessible only to NCM DWG members. This errata will be published once it gets approved.

Additional Details:

Mobile Broadband Interface Model Specification:
http://www.usb.org/developers/devclass_docs/MBIM10.zip

Mobile Broadband Driver Model Specification:
http://msdn.microsoft.com/library/windows/hardware/ff560543(v=VS.85).aspx

Device.Network.MobileBroadband.GSM.IdentityMorphing

Mobile broadband devices MUST support Identity Morphing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile broadband devices based on USB protocol must support Identity Morphing. Implementing this requirement in the device firmware enables the device manufacturers to take advantage of Microsoft's inbox MB Class Driver in Windows 8 and the flexibility of using their own driver for previous generations of Windows operating systems version 7 and below. Links to the relevant specifications are provided in the Additional Information section below.

Additional Information: Identity Morphing Specification: See MSDN.

Device.Network.MobileBroadband.GSM.Loopback

Mobile broadband devices based on USB protocol MUST implement loopback functionality for performance and payload conformance testing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile broadband devices based on the USB protocol must implement a loopback functionality as detailed in the specification in the device firmware. Note that Loopback functionality is only tested for the data packet as it is the one in the performance critical path. Devices must pass the loopback test for performance requirements. Specifically, devices must be able to support combined throughput of 100 Mbps (50 Mbps uplink / 50 Mbps downlink) or above and up to 5% loss rate.

Links to the relevant specifications are provided in the Additional Information section below.

Additional Information:

Loopback implementation guide for device firmware: http://msdn.microsoft.com/en-us/library/windows/hardware/hh975390.aspx

MB Miniport Driver Performance Requirements: http://msdn.microsoft.com/en-us/library/windows/hardware/ff557193(v=vs.85).aspx

Device.Network.MobileBroadband.GSM.MultiCarrierFunctionality

Mobile broadband devices that support multi-carrier feature must support the multi-carrier functionality.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile broadband devices that support multi-carrier feature must support the multi-carrier functionality and should also do the following:

Must meet the multi-carrier performance requirements available in the Mobile Broadband Driver Model Specification.

Must stay on the bus when changing home providers.

Must successfully pass all applicable logo tests covering all the different cellular class technologies that the device is capable of connecting to.

Mobile broadband devices supporting multi-carrier feature must meet the multi-carrier performance requirements specified in the mobile broadband driver model specification. Mobile broadband devices that support multi-carrier feature must not do a bus / device re-enumeration or power reset the device resulting in PnP re-enumeration to the Windows when changing the home providers. If the device is capable of supporting GSM and CDMA cellular class technologies, then the device must execute both GSM as well as CDMA logo tests. For this to be covered correctly, the location of logo

test execution must be in the coverage area of at least one GSM and one CDMA cellular class technologies.

Device.Network.MobileBroadband.GSM.MultiplePDPContext

Multiple PDP context support

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

With this feature, Windows apps can communicate over different PDP contexts (virtual channels) in mobile networks. Mobile Operators use different PDP contexts to create the differentiated experiences and innovative services. Third party app developers can use this feature to build great quality VOIP and video streaming experiences partnering with mobile operators.

Device firmware capable of multiple PDP contexts need to comply with the MBIM specification. Specifically, device firmware should support multiple IP data streams as detailed in section 10.5.12.1 in the MBIM specification. This includes supporting all the control implementation of CIDs and IP data streams for full support of multiple PDP contexts.

Device firmware must support a total of 8 dual bearer (IPv4 & IPv6) PDP contexts for usage by Windows.

This includes 1 for internet connectivity and 7 additional for Operator Apps.

Devices are not required to expose their internal, firmware managed PDP contexts used for SMS and any other administration context to Windows.

Device firmware should be able to leverage host OS request for a PDP context that is already device managed internally in its firmware to be handled gracefully.

Device firmware should continue to abstract SMS PDP contexts and route them through the SMS CIDs regardless of the bearer used underneath.

Device.Network.MobileBroadband.GSM.ReliableCSConnectivity

Wireless WAN device on systems that support Connected Standby must deliver reliable connectivity in Connected Standby.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The device seamlessly transitions between D0 and D3 warm states while in Connected Standby (CS). The device maintains both L2 & L3 connectivity while in CS. The device wakes up on matching wake patterns only. There are no spurious wakes while in CS. The wake packets are delivered without delay or buffering. RealTimeCommunication apps stay connected in CS over IPv4 and IPv6.

Device.Network.MobileBroadband.GSM.SupportFastDormancy

The GSM class of Mobile Broadband devices must support Fast Dormancy mechanism as defined by 3GPP in release 8.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile broadband devices must implement the fast dormancy mechanism as defined by 3GPP in revision 8. Fast Dormancy is a battery life savings mechanism for UE (User Equipment) devices that allows the devices to request the network to put them in a low power channel. UE sends a SIGNALLING CONNECTION RELEASE INDICATION (SCRI) message (sent by the UE to the network) with the IE "Signaling Connection Release Indication Cause" present and set to "UE Requested PS Data session end".

Device.Network.MobileBroadband.GSM.SupportUSBSelectiveSuspend

USB based Mobile Broadband devices must support Windows implementation of USB selective suspend.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB based Mobile Broadband devices must support Windows implementation of USB selective suspend (SS). No alternate USB SS implementation is allowed.

Device.Network.MobileBroadband.GSM.SupportWakeOnMB

Mobile broadband class of devices must support the following wake on mobile broadband capabilities.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Mobile broadband class of devices must support the following wake on mobile broadband capabilities.

- Devices MUST support 32 bitmap wake patterns of 128 byte each.
- Devices MUST wake the system on register state change.
- Devices MUST wake the system on media connect.
- Devices MUST wake the system on media disconnect.
- GSM and CDMA class of Devices MUST wake the system on receiving an incoming SMS message.
- Devices that support USSD MUST wake the system on receiving a USSD message.
- Devices MUST support wake packet indication. NIC should cache the packet causing the wake on hardware and pass it up when the OS is ready for receives.

Mobile broadband class of devices must support wake on mobile broadband. A device should wake the system on above mentioned events. Note that wake on USSD is mandatory only if the device reports that it supports USSD; otherwise, it is optional. See the following MSDN documentation for more information on the SMS and register state wake events.

- NDIS_STATUS_WWAN_REGISTER_STATE
- NDIS_STATUS_WWAN_SMS_RECEIVE

Device.Network.MobileBroadband.GSM.USSD

The GSM class of mobile broadband devices that implement Unstructured Supplementary Service Data (USSD) must support USSD based on the Mobile Broadband Driver Model.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows Mobile Broadband Driver Model is updated to include the full support of sending and receiving USSD messages. Devices that implement USSD must support USSD based on the Mobile Broadband Driver Model.

Send comments about this topic to Microsoft

Device.Network.Switch.Manageability

In this topic:

Device.Network.Switch.Manageability.NetworkSwitchProfile [If Implemented]

Device.Network.Switch.Manageability.NetworkSwitchProfile

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The proposed DTMF Network Switch Profile defines the CIM model and associated behavior for the management of a network switch (including the CIM classes, associations, indications, methods, and properties). It is not necessary that a datacenter network switch implements the full proposed DMTF Network Switch Profile, as only a subset of functionality is required to meet this manageability requirement. This subset must include the following management operations via implemented CIM classes remotely over WS-Man. This requirement is an If Implemented requirement and the functionality that needs to be met if this requirement is implemented is as follows:

- Get, Enable and Disable Switch Features
- Get, Enable and Disable Ports
- Set Port to Access or Trunk mode
- Get, Set Port Description
- Get a list of VLANs for a Trunk Port
- Get the VLAN for an Access Port
- Add/Remove a VLAN to/from a Trunk Port
- Get a list of VLANs
- Enable/Disable a VLAN
- Create/Delete a VLAN

- Change VLAN Name
- Shutdown/Restart Switch
- Get, Set Switch Host Name
- Get Firmware Version Info
- Get, Set Banner for login

Device.Network.Switch.Manageability.NetworkSwitchProfileView

This is a mandatory requirement for certification that the Layer-3 capable network switch must have the capability to be configured by the Windows PowerShell Desired State Configuration (DSC) mechanism.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Desired State Configuration (DSC) enables deploying and managing configuration data for services and managing the environment in which these services run. Additional information can be found at http://technet.microsoft.com/en-us/library/dn249912.aspx.

A datacenter Layer-3 network switch native OMI based provider implements a specific set of view classes that define the DSC experience and requirements. The design of the view classes provides a higher-level task-based abstraction compared to the full Network Switch CIM schema. The design of the view classes would be based on the CIM standard as defined by DMTF in this publication http://www.dmtf.org/sites/default/files/standards/documents/DSP0004_3.0.0.pdf.

This declarative way of configuring the switches ensures that multiple round trips are not required for the configuration and provides a vendor agnostic interface to achieve that. Additionally, it also ensures that the network switches do not deviate from this initial configuration, thereby improving the reliability of the deployment.

The approach with CIM view classes greatly simplifies the schema for the initial configuration and enables the above goals of configuration. The switch vendors have the flexibility to choose their own underlying implementation for satisfying the contract of the DSC provider represented by the view classes.

All categories of manageability operations would be validated as follows:

- The change request DSC document, a Managed Object Format File (MOF) file, is successfully received and consumed by the network switch. This document specifies the desired state into which the network switch must transition.

- Depending on the targeted functional area of the network switch, different methods are used to verify the intended state of the network switch. The related requirements call these out individually.

In addition to the consumption of DSC documents in the above manners, the switch also supports Enumerate and Get functionality on the configuration element(s) so that an explicit Enumerate or Get operation can be invoked on the switch to return the state information of the configuration elements.

A datacenter Layer-3 network switch provides this support natively without the aid of an external companion provider operating in a separate device enclosure.

Send comments about this topic to Microsoft

Device.Network.WLAN

Windows 10 WLAN drivers can implement either of the following driver models:

- WDI Driver Model introduced in Windows 10 (supports new Windows 10 features)
- Native Wi-Fi Driver Model (does not support new Windows 10 features)

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportConnectionToAP

In this topic:

Device.Network.WLAN.SupportConnectionToAP.ConnectionToAP (Mandatory)

Applies to	Windows 10 x64 Windows 10 x86

Description:

Devices must report supported band information for each respective category. In addition, device must comply with the following requirements.

Minimize CPU Utilization

The Wi-Fi device should conform to the following requirements for minimizing CPU utilization.

- While in EXTSTA mode and in D0 state, WLAN devices must not interrupt OS more than 1 time per data packet (non D0 coalesced packets only. If WLAN device supports D0 packet coalescing, WLAN device should comply with the D0 packet coalescing spec for D0 coalesced packets). Note that if WLAN device is using SDIO bus interface, interrupts generated by SDIO host controller per data packet are exempted from this requirement. Non data packet-related interrupts, if needed, must not exceed an average of 3 interrupts per second when measured over a 2 minute period. Also, as a best practice, in active state (when there is data traffic), the non-data packet related interrupts should be issued within 1 millisecond of a valid packet-related interrupt.
- In D0 state, all (if any) miniport specific periodic maintenance timers must be specified using an available "timer coalescing" API, with a minimum 2 second period and 1 second tolerance. This requirement will be tested in a long running connected state when there is no change in connectivity. All such timers must be cancelled in D3 state. Note that this requirement doesn't apply to timers defined in IEEE 802.11 specification e.g., connection timers such as association timers, roaming timers etc.
- An individual DPC (Deferred Procedural Call) duration MUST not exceed 2 milliseconds. Accumulated DPC duration should be less than 4 milliseconds over any 10 millisecond window.

In low power states, if the device is not Wake on Wireless capable, WLAN device must not interrupt the CPU. If the device is Wake on Wireless capable, WLAN device must not interrupt the CPU except on wake triggers indicated by NDIS for Wake on Wireless LAN. All interrupts not related to wake triggers must be cancelled.

Support Multiple Device Instances

Plug and Play can support multiple instances of a device. Multiple instances of the device can exist and function in the same system at the same instance. For network communications devices, the Plug and Play IDs and resource support MUST be sufficient to allow multiple network communications devices to be added automatically to the system. This requirement implies that all device resources MUST be set and read through the standard interfaces provided by the bus on which the device resides. For PCI devices, this interface is the PCI configuration space. Also, device parameter settings MUST be stored in the registry.

Comply with following specifications: NDIS 6.50, WDF/KMDF

Network device drivers must make only Network Driver Interface Specification (NDIS) 6.50 or Windows Driver Foundation (WDF) calls. Any calls to kernel mode components are not allowed.

Association Timing Requirement

On WPA2-PSK networks, WLAN devices must finish the association within **300ms**. It is measured as the time between the events when OS issues an OID "OID_WDI_TASK_CONNECT" and miniport sends an "NDIS_STATUS_WDI_INDICATION_ASSOCIATION_RESULT" indication to the OS.

Roam Timing Requirement

For devices that **do not** support Fast Roaming (this means that the STA does not support sending/receiving action frames nor utilitizes 802.11r Fast BSS Transitions. In addition, AP-side 802.11k Neighbor Reports and 802.11r is not enabled in the AP): Roaming action must complete within **100ms**. This time is measured starting when the OS issues the OID_WDI_TASK_CONNECT or OID_WDI_TASK_ROAM command and ends when the NDIS_STATUS_WDI_INDICATION_ASSOCIATION_RESULT notification is received by the OS with the AssociationStatus set to WDI_ASSOC_STATUS_SUCCESS .

For devices that **do** support Fast Roaming (this means that the STA does support sending/receiving action frames and utilizes 802.11r Fast BSS Transitions. In addition, AP-side 802.11k Neighbor Reports and 802.11r is enabled in the AP): Roaming action must complete within **50ms**. This time is measured starting when the OS issues the OID_WDI_TASK_CONNECT or OID_WDI_TASK_ROAM command and ends when the NDIS_STATUS_WDI_INDICATION_ASSOCIATION_RESULT notification is received by the OS with the AssociationStatus set to WDI_ASSOC_STATUS_SUCCESS.

Device Initialization Timing Requirement

WLAN devices must meet the following requirement during boot time, resume from hibernate, and device enable/disable actions.

- Device must complete the OpenAdapterHandler routine within 1 second. Time is measured between when NDIS calls OpenAdapterHandler function as part of a system PnP operation and OpenAdapterCompleteHandler(NDIS_STATUS_SUCCESS) is returned. In OpenAdapterHandler, the driver must have completed all its device initialization.

Radio State Change Timing Requirement

WLAN devices must be able to change the state of the radio within 2 sec. This duration will be measured between the request is sent through OID_WDI_TASK_SET_RADIO_STATE and when the miniport indicates NDIS_STATUS_WDI_INDICATION_RADIO_STATUS back.

Scan Preferred Channel Sets

Preferred channels set: When supported and permitted by the regulatory domain, the miniport driver must prefer the following channels when scanning for available networks or roaming to find a candidate access point:

- 2.4 Ghz channels: 1 to 14
- 5GHz U-NII Low channels: 36, 40, 44, 48
- 5GHz U-NII Mid channels: 52, 56, 60,64
- 5GHz U-NII Worldwide channels: 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140
- 5GHz U-NII Upper channels: 149, 153, 157, 161,165

The driver must report support for these and any other channels it supports through the ChannelList provided in the WDI_BAND_INFO of OID_WDI_GET_ADAPTER_CAPABILITIES.

Scan Action Requirement

WLAN device must start scanning when it receives a "OID_WDI_TASK_SCAN " from OS and reply back with an indication " NDIS_STATUS_WDI_INDICATION_SCAN_COMPLETE " to the OS as soon as it completes the scan. In case of active scanning, miniport is expected to send the active wildcard probes to the network channels to meet the scanning goals. In case of passive scanning, miniport is not expected to send any probes to the network channels.

This priority order should be followed for scanning.

- NLO hint channels
- Preferred channels
- Any remaining channels

The timings listed below will be measured from the time stamp when the device receives OID "OID_WDI_TASK_SCAN" from OS to the time stamp when the respective indication is provided to the OS by the miniport.

- If the Network list offload hints are available, the device should leverage the network list offload hints to optimize scanning behavior and return "NDIS_STATUS_WDI_INDICATION_NLO_DISCOVERY" indication when a matching profile is found within the following timings:
  - Scanning a network where active scanning is allowed - 20ms/channel
  - Scanning a network where only passive scanning is allowed - 120ms/channel
- If there is no match found using Network List Offload hints, WLAN device should next scan the preferred channels in the list above. For scanning the channels in the preferred channel list, WLAN device should not take more than 3.5 seconds (time includes scanning for both active and passive channels). The newly created list of surrounding BSS entries should be returned on the next BSS list query from the OS.
- If there is no match found in above 2 cases, WLAN device should next scan any remaining channels. WLAN device should not take more than 4 sec for the entire scan operation.

Scanning Behavior when Resuming from Sleep/Screen Off

When resuming from sleep/screen off, the WLAN device is expected to reconnect to the same AP that it was connected to before going to sleep, if it is available. WLAN devices must meet the following timings for detecting its presence:

- Reconnecting to a channel were active scanning is allowed - 50ms
- Reconnecting to a channel were only Passive scanning is allowed - 120ms

If the last connected network is not found, WLAN device must scan for networks in the priority order listed below.

- NLO hint channels
- Preferred channels
- Any remaining channels

WLAN device must follow the timing constraints as defined above in the general scanning section. The timings will be measured from the time stamp when the device reports as being in D0 state (OID_WDI_SET_POWER_STATE completing) to the time stamp when the respective indication is provided to the OS by the miniport.

Scan Non-Broadcast Networks

WLAN device must be able to scan for and report the presence of non-broadcast (i.e. Hidden) networks.

Support Coordinated Scanning (If-Implemented) (WDI drivers only)

VOIP Operations needs to be able to prioritize network resources such that data quality is maintained while still allowing for mobility. As such in the wireless stack, we have introduced the concept of Media Streaming Mode. This feature specifically calls out the key scanning implications / behavior when in Media Streaming Mode. Please refer to the WDI spec for more information.

Support Raw Action Frames (If-Implemented)(WDI drivers only)

Raw Action Frames are required for transmitting ANQP Query/Response frames, 802.11k Neighbor Report/Request Frames,and 802.11v BSS Transition Management Report/Request Frames. The device must report that it supports Raw Action Frames in WDI_GET_ADAPTER_CAPABILITIES. Please refer to the WDI Spec for implementation details. 802.11k Neighbor Reports and 802.11vBSS Transition Management Frames will be used to improve roaming performance in the operating system. By utilizing 802.11k Neighbor Reportsand 802.11v BSS Transition Management Frames, the OS has an improved understanding of its surroundings and allows for improved roaming performance, thus leading to less glitches for the end user when on a voice/video call.

Support 802.11r Fast BSS Transition (If-Implemented)(WDI drivers only)

802.11r Fast BSS Transitions allow for faster key exchange speeds, leading to improved roaming performance. The driver must support a new association type for Fast BSS Associations and be able to send the appropriate indications to the OS during the association/key exchange process. Please refer to the WDI spec from more information.

Low Latency Performance

WLAN devices should be able to support low latency applications/scenarios such as Wi-Fi Display/Lync/Skype. In order to do so, WLAN device must support the following:

- WMM should be supported on all ports including virtualized ports – ExtSTA, Wi-Fi Direct Role Port (Group Owner), and Wi-Fi Direct Role Port (Client).
- WMM should be supported on Wi-Fi Direct Role Port (Group Owner) and Wi-Fi Direct Role Port (Client) even if the AP connected via ExSTA port does not support WMM.
- The NIC should be capable of supporting prioritization across two ports (ExtSTA & one Wi-Fi Direct Role Port) at the same time.

- Devices should be able to prioritize traffic based on 802.11e Access Category (AC) tagging.
- When the NIC is virtualized with Concurrency in single channel, it should be able to prioritize transmit traffic across different virtual ports based on WMM & Media Streaming indication.
- When the NIC is virtualized with Concurrency across multiple channels, it should be able to prioritize transmit traffic across different virtual ports based on WMM and Media Streaming Indication and receiving traffic is serviced across the concurrent channels.

When WMM prioritization is used with AC_VI or AC_VO (Voice/Video), WLAN device must meet the following latency and packet loss requirements.

	ExSTA	Wi-Fi Direct Role Port	Max. One Way Latency at UDP for AC_VI/AC_VO Traffic	Packet Loss for AC_VI/AC_VO Traffic
ExSTA Only	AC_VI/AC_VO	NA	30ms	0.5%, not more that 3 consequetive packets
ExSTA + Wi-Fi Direct in Single Channel Concurrency	AC_VI/AC_VO		30ms	0.5%, not more that 3 consequetive packets
		AC_VI/AC_VO	30ms	0.5%, not more that 3 consequetive packets
	AC_VI/AC_VO	AC_VI/AC_VO	30ms	0.5%, not more that 3 consequetive packets
ExSTA + Wi-Fi Direct in Multi Channel Concurrency	AC_VI/AC_VO		40ms	0.5%, not more that 3 consequetive packets
		AC_VI/AC_VO	40ms	0.5%, not more that 3 consequetive packets

	AC_VI/AC_VO	AC_VI/AC_VO	50ms	0.5%, not more that 3 consequetive packets

Minimum Throughput Performance

Throughput With Concurrent Operation: The 802.11 devices must not have more than 20% aggregate throughput degradation when data flow is divided among 3 ports (with and without Multi-Channel/Band operation) namely – ExtSTA, Wi-Fi Direct Role Port (GO) and Wi-Fi Direct Role Port (Client) compared with aggregate throughput when only one Wi-Fi port is connected.

Throughput:

This requirement for throughput is applicable to all types of ports (ExtSTA, Wi-Fi Direct Role Port (GO) and Wi-Fi Direct Role Port (Client)) individually.

Device must be capable of supporting the following throughput numbers over a TCP connection for a particular combination of Phy type, number of streams and channel width. Throughput is measured at the application layer using NTTTCP perf measurement tool built into Windows hardware certification kit.

- If WLAN device supports 802.11ac Phy Type (Note that 802.11ac Phy Type support is optional for WLAN devices. 802.11ac devices will also be tested for 802.11n operations)
    - 802.11ac combinations will be tested on 5 Ghz band only.
    - Both transmit and receive side throughput will be tested. Device is expected to meet same throughput number (listed below) for a particular combination on both transmission and reception side.

Max supported spatial stream by the NIC will be tested. E.g., if the NIC supports 3 spatial streams, combinations with spatial stream 3 will be tested.

Channel Width	20 Mhz		40 Mhz		80 Mhz	
# of Spatial Streams	Max Phy rate (100%)	Throughput expected by Windows Certification	Max Phy rate (100%)	Throughput expected by Windows Certification	Max Phy rate (100%)	Throughput expected by Windows Certification
1 Stream	86	18	200	42	433	90
2 Stream	173	38	400	88	866	191

3 Stream or higher*	258	47	600	110	1300	237

*All speeds are in Mbps.

* Combinations not required for SDIO and USB 2.0 Bus Type devices. These devices with support for more than 2 streams must be able to attain values listed for 2 streams. Note that USB 3.0 devices are not excluded.

- 802.11n WLAN devices
  - 802.11n combinations will be tested over both 2.4 Ghz and 5 Ghz bands if both bands are supported on the device.
  - Both transmit and receive side throughput will be tested. Device is expected to meet same throughput number (listed below) for a particular combination on both transmission and reception side.
  - Max supported spatial streams by the NIC will be tested. E.g., if the NIC supports 3 spatial streams, combinations with spatial streams 3 will be tested.

Channel Width	20 Mhz		40 Mhz	
# of Spatial Streams	Max Phy rate (100%)	Throughput expected by Windows Certification	Max Phy rate (100%)	Throughput expected by Windows Certification
1 Stream	72	15	150	31
2 Stream	144	32	300	66
3 Stream or higher*	216	40	450	82

*All speeds are in Mbps.

*Combinations not required for SDIO Bus Type devices. SDIO devices with support for more than 2 streams must be able to attain values listed for 2 streams.

Wi-Fi Power-Save Mode

The Wi-Fi driver is required to perform detection and negotiation of proper Wi-Fi Power Save Mode (PSM) between the device and the Wi-Fi Access Point and report it as AutoPowerSaveMode in the WDI_STATION_ATTRIBUTES. If the driver reports that it supports PSM detection, WLAN service will delegate the PSM decision to the driver by default.

If the driver supports the AUTO-PSM capability the Wi-Fi service will no longer set the broadcast management filter to receive beacons from the miniport and will instead set an OID to turn on Auto-PSM in the driver. When in Auto-PSM, the driver should always negotiate PSM mode when it detects that the AP supports it and manage the use of PSM between the device and the Wi-Fi Access Point to ensure optimal connectivity while using the least amount of power.

Connection Loss

All WLAN devices must detect and indicate the loss of the connected AP (no beacons) within 20 beacon intervals.

Permit Information Elements to be added to Association Frames (Request/Response)

All 802.11 devices must permit Information Elements to be added to association frames, both requests and responses. This includes adding currently specified Information Elements, such as Wi-Fi Protected Setup as well as other vendor extended Information Elements.

Support Filtering for 32 Multicast Addresses

WLAN hardware must support at least 32 multicast addresses on each port. Both STA and Wi-Fi-Direct ports need to support filtering 32 multicast addresses separately.

Support Separate Beacon and Probe Information Elements

All 802.11 devices must separately indicate the Wi-Fi Protected Setup IEs that are received in Beacon frames and probe-response frames. If the device has received both a beacon frame and a probe-response frame from a particular BSSID, then it must provide two instances of the IE, where one instance is the most recently received WPS IE from the Beacon and one instance is the most recently received WPS IE from the probe response.

Transmit Packets on Any Boundary

Buffer alignment refers to whether a buffer begins on an odd-byte, word, double-word, or other boundary. Devices must be able to transmit packets with any of the packets fragments beginning on an odd-byte boundary. For performance reasons, packets must be received into contiguous buffers on a double-word boundary.

TCP Checksum Offload (If-Implemented) (WDI drivers only)

TCP/IP protocol is designed to provide a reliable data transfer mechanism. This calls for the TCP stack to implement various protection mechanisms to detect and fix such errors. One such mechanism is the concept of using "checksums" to ensure reliability. In Windows we do the checksum computations in the OS, request retries of packets and manage the packets within the HOST. The WLAN Device must report that it supports offloads and be able to offload in the OS-prescribed manner. Device must be able to detect if the offload is causing performance issues and be able to disable the offload. Offloads should work across Station Connectivity and Wi-Fi Direct usage scenarios.

Connectivity Types

All WLAN devices must be able to connect using the following Wi-Fi connection types. All WLAN devices declare support for these through WDI_ALGO_PAIRS reported in OID_WDI_GET_ADAPTER_CAPABILITIES

Authentication and cipher support required in infrastructure mode:

Authnetication	Cipher
Open	None
Open	WEP-40bit
Open	WEP-104bit
Open	WEP
WPA-Enterprise	TKIP
WPA-Enterprise	CCMP
WPA-Personal	TKIP
WPA-Personal	CCMP
WPA2-Enterprise	TKIP
WPA2-Enterprise	CCMP
WPA2-Personal	TKIP
WPA2-Personal	CCMP

[Native Wi-Fi Drivers Only] (If-Implemented) Authentication and cipher support required in ad-hoc (IBSS) mode:

Authnetication	Cipher
Open	None
Open	WEP-40bit

Open	WEP-104bit
Open	WEP
WPA2-Personal	CCMP

Support Hibernate and Resume

If the device supports Hibernate and Resume, the device should not fall off the bus as part of its initial (boot time) firmware download process or while going to hibernate state.

Support Radio Management (Radio On/Off)

Device must support Radio Management requests from the OS via the WDI_TASK_SET_RADIO_STATE command. Please refer to the WDI spec for more information.

Support Wi-Fi Protected Setup (WPS)

Devices must support Wi-Fi Protected Setup. OS will issue WDI_SET_P2P_WPS_ENABLED command to indicate to the LE when WPS is set. Please refer to the WDI Spec for more information.

Support Packet Filters (If-Implemented)

Packet Filters allow the OS to filter different types of packets when upon receipt. The OS uses the WDI_TLV_PACKET_FILTER_PARAMETERS TLV to indicate which type of packet to filter for. Please refer to the WDI spec for more information.

Obtain WFA Certifications (if-Implemented)

It is strongly recommended that devices obtain the following WFA Certifications:

- 802.11n
- WMM
- WPA2Protected Management Frames (PMF)

Implement 802.11a (If-Implemented)

Device must report 802.11a PHY type 4 (WDI_PHY_TYPE_OFDM) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Implement 802.11b (Mandatory)

Device must report 802.11b PHY type 2 (WDI_PHY_TYPE_HRDSS) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Implement 802.11g (If-Implemented)

Device must report 802.11g PHY type 5 (WDI_PHY_TYPE_ERP) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Implement 802.11n (If-Implemented)

Recommended for optimal WLAN performance.

Device must report 802.11n PHY type 7 (WDI_PHY_TYPE_HT) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Implement 802.11ac (If-Implemented)

Device must report 802.11ac PHY type 8 (WDI_PHY_TYPE_VHT) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Support Hang Detection and Recovery (If-Implemented)(WDI drivers only)

Wi-Fi device firmware has been known to hang and / or get in a stuck state. Once that happens, the Lower Edge driver would either crash causing a 9F (Blue Screen) or the Wi-Fi subsystem gets into a state which requires a system reboot for the device to be functional again. In either case, the user is faced with a negative experience in their connectivity and their general system usage is disrupted. As an integral part of WDI, we have designed a mechanism to detect when the firmware gets into these states and recover the device seamlessly. This will ensure that user will see a minimal disruption in service by ensuring that the Wi-Fi device stack recovers and resumes connectivity to the network without the system needing a reboot. Devices must report support for Hang Detection and Recovery in WDI_GET_ADAPTER_CAPABILITIES. Please refer to the WDI Spec for implementation details.

Requirement – Hardware / Firmware

System ACPI firmware: The System will provide the ACPI methods to PDLR the device either at a bus or at the device level.

System hardware: The system will allow for a PDLR (full device level reset). All systems must support PDLR.

System: The System will indicate support for PDLR support.

Device: The Lower Edge driver will be able to gather dumps with 25 ms and 250 Kb size

System: The system must complete the reset within 10 seconds.

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportDot11W

In this topic:

Device.Network.WLAN.SupportDot11W.Dot11W (If-Implemented)

Applies to	Windows 10 x64 Windows 10 x86

Description:

Device must report support for 802.11w Protected Management Frames in WDI_STATION_ATTRIBUTES. IEEE 802.11w is an addition to IEEE 802.11 suite of standards to enhance the security of management frames. Please refer to the WDI Spec for implementation details.

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportFIPS

In this topic:

Device.Network.WLAN.SupportFIPS.FIPS (If-Implemented)

Applies to	Windows 10 x64 Windows 10 x86

Description:

Report support for FIPS in WDI_STATION_ATTRIBUTES. Please refer to the WDI Spec for implementation details. The FIPS (Federal Information Processing Standards) is a United States government requirement for use within communication and computer systems.

Your WLAN device must support one of the two following solutions for FIPS support:

- Hardware based solution: you may choose to support FIPS directly through your hardware device. You can do so by declaring DOT11_EXTSTA_ATTRIBUTES_SAFEMODE_CERTIFIED(for NWIFI drivers) and WDI_STATION_ATTRIBUTES – Host FIPS Mode Implemented (for WDI Drivers)in your driver. If you declare hardware based support for FIPS you the device manufacturer are responsible for ensuring compliance with the FIPS standard. Windows cannot and will not test adherence to the FIPS standard in this mode.
- Software based solution: Windows implements a software based solution that conforms to the FIPS standard. You can do so by declaring DOT11_EXTSTA_ATTRIBUTES_SAFEMODE_OID_SUPPORTED, and implementing OID_DOT11_SAFE_MODE_ENABLED or OID_DOT11_SAFE_MODE_HT_ENABLED in your driver.

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportHostedNetwork

In this topic:

Device.Network.WLAN.SupportHostedNetwork.HostedNetwork (If-Implemented) (Native Wi-Fi drivers only)

Applies to	Windows 10 x64 Windows 10 x86

Description:

With this feature, a Windows computer can use a single physical wireless adapter to connect as a client to a hardware access point (AP), while at the same time acting as a software AP allowing other wireless-capable devices to connect to it.

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportHotspot2Dot0

In this topic:

Device.Network.WLAN.SupportHotspot2Dot0.Hotspot2Dot0 (If Implemented) (WDI drivers only)

Applies to	Windows 10 x64 Windows 10 x86

Description:

Device must report that it supports ActionFrameSupport in WDI_TLV_INTERFACE_CAPABILITIES and optinally report that it supports ConnectToSSIDsBelongingToHESSID in WDI_STATION_ATTRIBUTES. Please refer to the WDI Spec for implementation details.

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportMACAddressRandomization

In this topic:

Device.Network.WLAN.SupportMACAddressRandomization.MACAddressRandomization (If-Implemented) (WDI drivers only)

Applies to	Windows 10 x64 Windows 10 x86

Description:

The Wi-Fi MAC address is currently used to track users as they move through public spaces, from hot-spot to hot-spot, and even within a department store or a mall. By randomizing the MAC address, Windows Phones users cannot be tracked without their consent. To support this feature, the device must report to the operating system that it supports MAC Address Randomization via WDI_GET_ADAPTER_CAPABILITIES. Please refer to the WDI spec for feature implementation details.

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportWakeFromLowPower

In this topic:

Device.Network.WLAN.SupportWakeFromLowPower.WakeFromLowPower (If-Implemented)

Applies to	Windows 10 x64 Windows 10 x86

Implement Wake on WLAN

Description:

WLAN devices that implement WoWLAN (Wake on Wireless LAN) must support all the features related to WoWLAN capability. Implementation of subset of below features will not be considered for certification. WLAN devices should do the following:

- Must indicate the specific Wake on Wireless LAN (WoWLAN) capability that is supported.
- Must support at least 22 WoWLAN bitmap wake-up patterns of 128 byte each.
- Must be able to perform GTK (WPA/WPA2) and IGTK refresh (WPA2) while in the Dx state.
- Must support wake on GTK and IGTK handshake error.
- Must support wake when 802.1x EAP-Request/Identity Packet is received.
- Must support wake when four way handshake requests is received.
- Must support wake on association lost with current AP.
- Must support wake when a network matches NLO (Network list offload) hints.
- Must support wake packet indication. NIC should cache the packet causing the wake on hardware and pass it up when the OS is ready for receive.
- Must support ARP and NS offloads to ensure link local network discovery. WLAN device should be able to respond to ARP and NS requests without interrupting the CPU when the device is in low power (D3) state. Devices must support at least 1 ARP offload and at least 2 NS offloads (each NS offload with up to 2 target IPv6 addresses).

Network List Offload

Description:

WLAN devices should support 10 offloaded BSSID profiles. Wi-Fi profiles that are marked auto-connect will be offloaded by the OS to the device/driver. The device should use the network list offload as a hint to optimize scanning behavior and return "NDIS_STATUS_WDI_INDICATION_NLO_DISCOVERY " indication when a matching profile is found.

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportWiFiDirect

In this topic:

Device.Network.WLAN.SupportWiFiDirect.WiFiDirect (If-Implemented)

Applies to	Windows 10 x64 Windows 10 x86

Support Wi-Fi Direct Base Functionality

Description

If implemented, support for Wi-Fi Direct by the Wi-Fi Driver to enable Miracast, Public APIs for Wi-Fi Direct to allow pairing to & from the PC, Accepting and Connecting to other Wi-Fi Direct Device for the GO & the Client Role. This includes support for concurrent operation over Wi-Fi Direct & Station.

Details

Device must report the relevant capabilities in the WIFI_GET_ADAPTER_CAPABILITIES per the WDI Specification

- WIFI_VIRTUALIZATION_ATTRIBUTES
- WIFI_P2P_ATTRIBUTES
- WIFI_BAND_INFO
- WIFI_PHY_INFO

If this requirements is supported, then the device must be capable of performing the following action as defined in the WDI Spec

- Scan for Wi-Fi Direct Devices & GO
- Be Discoverable as a Wi-Fi Direct Device & GO
- Pair with Wi-Fi Direct Devices & GO
- Accept Incoming Pairing from Wi-Fi Direct Devices & GO
- Connect to a Wi-Fi Direct Client or GO
- Accept Incoming Connections from a Wi-Fi Direct Client or a GO
- Support for Extensible IEs in During Pairing & (Re) Association

- Support for the providing the Link Quality & Throughput rates for the Wi-Fi Direct Connection irrespective of Role
- Support for indication of Operating Channel whether in GO or Client Role via the WIFI_INDICATION_LINK_STATE_CHANGE

Support Background Discovery

Description

If implemented**,** support for background discovery of Wi-Fi Direct Devices, this allows for applications that use the Wi-Fi Direct API to offload the discovery to enable a richer set of scenarios by allowing application to get a notification when a device is found vs. having to scan for them periodically. Support for this feature also enables better resource utilization.

Details

Device must report the relevant capabilities in the WIFI_GET_ADAPTER_CAPABILITIES per the WDI Specification

- WIFI_VIRTUALIZATION_ATTRIBUTES
- WIFI_P2P_ATTRIBUTES

If the device supports this requirement, it must be able to scan for Wi-Fi Direct Devices & GO on all supports periodically in D0 state.

Support eCSA

Description

If implemented**,** support for enhanced Channel Switch Annoucement (eCSA) as per the 802.11 specification, in the Wi-Fi Direct Client and in the Wi-Fi Direct GO. This will enable better performance by NIC moving the GO or responding to eCSA requests from the GO that is connected to.

Details

Device must report the relevant capabilities in the WIFI_GET_ADAPTER_CAPABILITIES per the WDI Specification

- WIFI_INTERFACE_ATTRIBUTES

If this requirements is supported then all Wi-Fi Direct Ports must support eCSA. If the Device is in GO Mode then it must be able to detect if the remote device (that it is connected to) supports eCSA and use eCSA to move to single channel configuration . If the Device is in client mode, then respond to eCSA requests from the remote device (that it is connected to). The Device must report change of Operating Channel for both Client & GO via the WIFI_INDICATION_P2P_GROUP_OPERATING_CHANNEL Indication.

OS will not control the eCSA behavior of the Device, this is device specific implementation.

Support Concurrency

Description

If implemented, support for operation of scenarios such as Miracast or Wi-Fi Direct & Wi-Fi Direct Services along with connectivity to the Internet. Without support for this, the device will only be able to connect to one end-point - Example cannot connect to Miracast while also being connected to the Internet.

Details

Device must report the relevant capabilities in the WIFI_GET_ADAPTER_CAPABILITIES per the WDI Specification

- WIFI_VIRTUALIZATION_ATTRIBUTES
- WIFI_P2P_ATTRIBUTES

If this requirements is supported, then Device must be capable of support at least 2 Wi-Fi Direct role ports concurrently in the following configuration in addition to the Wi-Fi Direct Device Port :

- A Group Owner (GO) on one Wi-Fi Direct Port and Client on the other Wi-Fi Direct port(s) Concurrently
- A Client on each Wi-Fi Direct Port Concurrently
- Device can support concurrently with Infrastructure connectivity on different channels across different bands (2.4/5 GHz) with a 2 concurrent channels.

Sample of Expected Supported Matrix for 2 Channel & 2 Port Concurrency

- If the WLAN Device Supports 802.11ac then :

	ExTSTA Port			**Wi-Fi Direct Role Port 1**			**Wi-Fi Direct Role Port 2**		

Case #	802.11n	802.11n	802.11ac	802.11n	802.11n	802.11ac	802.11n	802.11n	802.11ac
	2.4 Ghz	5 Ghz	5 GHz	2.4 Ghz	5 Ghz	5 GHz	2.4 Ghz	5 Ghz	5 GHz
1	STA	x	x	GO	x	x	x	x	x
2	STA	x	x	Client	x	x	x	x	x
4	STA	x	x	x	Client	x	x	x	x
6	STA	x	x	x	x	Client	x	x	x
7	STA	x	x	GO	x	x	Client	x	x
8	STA	x	x	Client	x	x	Client	x	x
9	STA	x	x	GO	x	x	x	Client	x
10	STA	x	x	Client	x	x	x	Client	x
11	STA	x	x	GO	x	x	x	x	Client
12	STA	x	x	Client	x	x	x	x	Client
14	STA	x	x	x	Client	x	x	Client	x
16	STA	x	x	x	Client	x	x	x	Client
18	STA	x	x	x	x	Client	x	x	Client
19	x	STA	x	GO	x	x	x	x	x
20	x	STA	x	Client	x	x	x	x	x
21	x	STA	x	x	GO	x	x	x	x
22	x	STA	x	x	Client	x	x	x	x
23	x	STA	x	x	x	GO	x	x	x

24	x	STA	x	x	x	Client	x	x	x
25	x	STA	x	GO	x	x	Client	x	x
26	x	STA	x	Client	x	x	Client	x	x
27	x	STA	x	GO	x	x	x	Client	x
28	x	STA	x	Client	x	x	x	Client	x
29	x	STA	x	GO	x	x	x	x	Client
30	x	STA	x	Client	x	x	x	x	Client
31	x	STA	x	x	GO	x	x	Client	x
32	x	STA	x	x	Client	x	x	Client	x
33	x	STA	x	x	GO	x	x	x	Client
34	x	STA	x	x	Client	x	x	x	Client
35	x	STA	x	x	x	GO	x	x	Client
36	x	STA	x	x	x	Client	x	x	Client
37	x	x	STA	GO	x	x	x	x	x
38	x	x	STA	Client	x	x	x	x	x
39	x	x	STA	x	GO	x	x	x	x
40	x	x	STA	x	Client	x	x	x	x
41	x	x	STA	x	x	GO	x	x	x
42	x	x	STA	x	x	Client	x	x	x
43	x	x	STA	GO	x	x	Client	x	x

44	x	x	STA	Client	x	x	Client	x	x
45	x	x	STA	GO	x	x	x	Client	x
46	x	x	STA	Client	x	x	x	Client	x
47	x	x	STA	GO	x	x	x	x	Client
48	x	x	STA	Client	x	x	x	x	Client
49	x	x	STA	x	GO	x	x	Client	x
50	x	x	STA	x	Client	x	x	Client	x
51	x	x	STA	x	GO	x	x	x	Client
52	x	x	STA	x	Client	x	x	x	Client
53	x	x	STA	x	x	GO	x	x	Client
54	x	x	STA	x	x	Client	x	x	Client
55	x	x	x	GO	x	x	Client	x	x
56	x	x	x	Client	x	x	Client	x	x
57	x	x	x	GO	x	x	x	Client	x
58	x	x	x	Client	x	x	x	Client	x
59	x	x	x	GO	x	x	x	x	Client
60	x	x	x	Client	x	x	x	x	Client
62	x	x	x	x	Client	x	x	Client	x
63	x	x	x	x	GO	x	x	x	Client
64	x	x	x	x	Client	x	x	x	Client

66	x	x	x	x	x	Client	x	x	Client

- If WLAN device does not support 802.11ac :

Case #	**Infra Port**		**WFD Port 1**		**WFD Port 2**	
	2.4 Ghz	5 Ghz	2.4 Ghz	5 Ghz	2.4 Ghz	5 Ghz
1	STA	x	GO	x	x	X
2	STA	x	Client	x	x	X
4	STA	x	x	Client	x	X
5	STA	x	GO	x	Client	X
6	STA	x	Client	x	Client	X
7	STA	x	GO	x	x	Client
8	STA	x	Client	x	x	Client
10	STA	x	x	Client	x	Client
11	x	STA	GO	x	x	x
12	x	STA	Client	x	x	x
13	x	STA	x	GO	x	x
14	x	STA	x	Client	x	x
15	x	STA	GO	x	Client	x
16	x	STA	Client	x	Client	x
17	x	STA	GO	x	x	Client

18	x	STA	Client	x	x	Client
19	x	STA	x	GO	x	Client
20	x	STA	x	Client	x	Client
21	x	x	GO	x	Client	x
22	x	x	Client	x	Client	x
23	x	x	GO	x	x	Client
24	x	x	Client	x	x	Client
26	x	x	x	Client	x	Client

Support Internet Sharing

Description

If implemented**,** support for Tethering over Virtualized Wi-Fi Port including support for Custom SSID and WPA2 Auth/Encryption for concurrent operation of tethering and station connectivity.

Devices are recommended to support tethering for a minimum of 3 clients.

Device must report the relevant capabilities in the WIFI_GET_ADAPTER_CAPABILITIES per the WDI Specification

- WIFI_VIRTUALIZATION_ATTRIBUTES
- WIFI_P2P_ATTRIBUTES
- WIFI_BAND_INFO
- WIFI_PHY_INFO

Support Advanced WMM Levels across WFD Role Ports

Description

This requirements captures the work needed to be done by the Wi-Fi Driver. This includes support for concurrent operation over Miracast, Station & Bluetooth.

Details

The Connection Quality & Performance of Wi-Fi Direct is essential to the Miracast Scenario. The assessment for the Device/Driver will provide the results for your device, below is guidance for a reasonable quality Miracast Connection.

WMM shall be supported on all ports including virtualized ports – ExtSTA, Wi-Fi Direct Role Port (Group Owner), and Wi-Fi Direct Role Port (Client).

- The NIC should be capable of supporting prioritization across two ports (ExtSTA & one Wi-Fi Direct Role Port) at the same time.
- Prioritize traffic based on 802.11e Access Category (AC) tagging.
- When the NIC is virtualized with Concurrency in single channel, it must be able to prioritize transmit traffic across different virtual ports based on WMM & Media Streaming indication.
- When the NIC is virtualized with Concurrency across multiple channels, it must be able to prioritize transmit traffic across different virtual ports based on WMM and Media Streaming Indication and receiving traffic is serviced across the concurrent channels.

When WMM prioritization is used with AC_VI or AC_VO (Voice/Video), WLAN device should meet the following latency and packet loss requirements.

	ExSTA	**Wi-Fi Direct Role Port**	**Max. One Way Latency at UDP for AC_VI/AC_VO Traffic**	**Packet Loss for AC_VI/AC_VO Traffic**
ExSTA Only	AC_VI/AC_VO	NA	30ms	0.5%, not more than 3 consecutive packets
ExSTA + Wi-Fi Direct in Single Channel Concurrency	AC_VI/AC_VO		30ms	0.5%, not more than 3 consecutive packets
		AC_VI/AC_VO	30ms	0.5%, not more than 3 consecutive packets
	AC_VI/AC_VO	AC_VI/AC_VO	30ms	0.5%, not more than 3

				consecutive packets
ExSTA + Wi-Fi Direct in Multi Channel Concurrency	AC_VI/AC_VO		40ms	0.5%, not more than 3 consecutive packets
		AC_VI/AC_VO	40ms	0.5%, not more than 3 consecutive packets
	AC_VI/AC_VO	AC_VI/AC_VO	50ms	0.5%, not more than 3 consecutive packets

Send comments about this topic to Microsoft

Device.Network.WLAN.SupportWiFiDirectServices

In this topic:

Device.Network.WLAN.SupportWiFiDirect.WiFiDirectServices (If-Implemented)(WDI drivers only)

Applies to	Windows 10 x64 Windows 10 x86

Description

If implemented**,** support for Wi-Fi Direct by the Wi-Fi Driver to enable Wi-Fi Direct Services for both subscribing to and publishing Wi-Fi Direct Services. Windows SDK will provide APIs for using Wi-Fi Direct Services such that app developers can build peer to peer applications for example: Multiplayer Games, File Sharing or Sync with Device

Details

Device must report the following capabilities per the WDI Specification:

- Action Frames Supported in WDI_INTERFACE_ATTRIBUTES
- Service Discovery Supported in WDI_P2P_ATTRIBUTES

P2P Service Names Discovery Supported in WDI_P2P_ATTRIBUTESMax P2P Service Name Advertisement Bytes Supported in WDI_P2P_ATTRIBUTESOptional Support is recommended for the capabilities listed below:

- Background Discovery Of P2P Devices And Services in WDI_P2P_ATTRIBUTES (Highly Recommended)
- Passive Availability Listen State in WDI_P2P_ATTRIBUTES (Highly Recommended)
- P2P Service Information Discovery Supported in WDI_P2P_ATTRIBUTES
- Max P2P Service Information Advertisement Bytes Supported in WDI_P2P_ATTRIBUTES

If this requirement is supported, then the device must be capable of performing the following action as defined in the WDI Spec:

- Support for Wi-Fi Direct
- Support for Sending both Request and Response Action Frames
- Support for Querying AQNP
- Support for Indication of received Action Frames

Send comments about this topic to Microsoft

Device.Portable.Core

Core

In this topic:

- Device.Portable.Core.MTPFunctionality
- Device.Portable.Core.MTPMultiSession
- Device.Portable.Core.MTPObjectProperties
- Device.Portable.Core.MTPStreams
- Device.Portable.Core.TransportBluetooth
- Device.Portable.Core.TransportIP
- Device.Portable.Core.TransportIPDLNA
- Device.Portable.Core.TransportUSB
- Device.Portable.Core.VideoCodec

Device.Portable.Core.AudioCodec

If a portable device can capture audio content, it must do so using a format supported natively in Windows.

Applies to	Windows 10 x64 Windows 10 x86

Description

If a portable device can capture audio content, it must do so in a format that Windows understands natively.

The following tables represent the list of in-box formats that Windows will render to; this is not the exhaustive of supported formats in Windows. The full list of formats supported can be found at: http://go.microsoft.com/fwlink/p/?linkid=242995&clcid=0x409.

Note that for a given format in the tables below it is not required to support all given Bit rates and Sample rates, it is however required to support a format that lines up within the ranges provided.

MP4 Audio Content

	Setting	Requirement
AAC	Codec	AAC Standard, AAC Profile level 2 minimum (0x29), compatible with (0x29, 0x2A, 0x2B, 0x2C, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33)
	bit rate for CBR	96, 128, 160, 192 Kbps
	Sample rate	44.1 or 48 kHz

	Channels	2

MP3 Audio Content

	Setting	Requirement
MP3	Codec	Version1, Layer3
	Bit rate for CBR files	From 32 to 320 kilobits per second (Kbps)
	Sample rate	16, 22.05, 24, 32, 44.1, 48 kilohertz (kHz)
	Channels	2

WMA Audio Content

	Setting	Requirement
WMA Standard	Codec	WMA9 or later
	Bit rate for CBR files	From 32 to 256 kilobits per second (Kbps)
	Average bit rate for VBR files	From 48 to 160 Kbps
	Maximum peak bit rate for VBR files	256 Kbps
	Sample rate	44.1 or 48 kilohertz (kHz)
	Channels	2

Device.Portable.Core.CustomDeviceServices

A portable device that implements custom MTP Services must meet the requirements defined in the MTP Devices Services Extension Specification.

Applies to	Windows 10 x64 Windows 10 x86

Description

The MTP Device Services Extension to the Media Transfer Protocol (MTP) helps an MTP Initiator, in this case Windows, find and access certain types of content stored on the device. Extension mechanisms have been defined that provide greater flexibility for applications that deal with specific content types defined by the device. These mechanisms provide greater extensibility than the existing data code mechanisms currently defined by the Media Transfer Protocol Specification, Revision 1.0.

If the portable device supports a custom device service, the implementation must have a valid ServiceInfo dataset, a valid ServiceCapabilities dataset, and a valid ServicePropertiesDesc dataset as defined in the MTP Device Services Extension Specification. All mandatory properties defined in the specification must be supported in the custom service.

The following table is a list of operations that are required when implementing MTP Services:

Operation	MTP Datacode	**Description**
GetServiceIDs	0x9301	This operation returns an array of ServiceIDs.
GetServiceInfo	0x9302	This operation returns the ServiceInfo dataset for a service.
GetServiceCapabilities	0x9303	All object format and method format information is reported by using the GetServiceCapabilities operation.
GetServicePropDesc	0x9304	This operation returns theServicePropertyDesc dataset for a service.
GetServicePropList	0x9305	This operation is similar to GetObjectPropList in the MTP specification, Revision 1.0. GetServicePropList reads properties from a service.
SetServicePropList	0x9306	This operation sets a ServiceProperty by using the ServicePropList dataset. It enables the writing of property values to a service.

UpdateObjectPropList	0x9307	This operation sets the property list for a particular object that will be updated with a new binary object. This operation can be used to replace the binary data of an existing object.
DeleteObjectPropList	0x9308	This operation removes the properties that are specified in the DeleteObjectPropList dataset from the specified object or objects.
DeleteServicePropList	0x9309	This operation removes the properties that are specified in the DeleteServicePropList dataset from the specified service.

If scenarios that can be implemented using capabilities defined in the MTP specification are not implemented using the operations, device properties, etc. defined in the MTP 1.0 specification (or later), then the device must define these scenarios in terms of device services and must support these services according to the requirements defined in the MTP Device Services Extension Specification. For example, a media exchange service may be defined to manage media synchronization between the initiator and responder that replaces functionality supported by standard MTP 1.0 behavior.

To expose a custom device service in Device Stage, a custom task must be authored and defined as part of the Device Stage authoring process. This is described in the Microsoft Device Experience Development Kit.

Design Notes:

- Refer to the MTP Device Services Extension Specification available at http://msdn.microsoft.com/en-us/windows/hardware/gg463541.aspx.
- Refer to the Microsoft Device Experience Development Kit available at http://msdn.microsoft.com/en-us/windows/hardware/gg463154.aspx.

Device.Portable.Core.DeviceServices

Portable devices that support defined MTP Services must implement these services according to the requirements defined in the MTP Device Services for Windows Specification.

Applies to	Windows 10 x64 Windows 10 x86

Description

The MTP Device Services architecture enables Windows to locate and use various services and content types located on a device. By creating extensions to the WPD API and MTP protocol, Windows can locate, consume, and interact with useful content and services on the device.

Supported Personal Information Management (PIM) Services:

- Contact Service
- Calendar Service
- Notes Service
- Tasks Service

Other Supported Services:

- Status Service
- Hints Service
- Device Metadata Service
- Ringtones Service
- SMS Service

If the device supports one or more of the PIM services, it must also support one of two synchronization services to actually synchronize the data. Windows supports the following in-box:

- Enumeration Sync
- Anchor Sync

It is up to the manufacturer to choose the services that is best suited for the device.

Vendors who choose to support custom Device Stage metadata packages, should ensure that the appropriate tasks or device properties are exposed correctly within the package.

For services to be accessible by a service aware initiator, the following service related operations must also be supported by the device:

Operation	MTP Datacode	**Description**
GetServiceIDs	0x9301	This operation returns an array of ServiceIDs.
GetServiceInfo	0x9302	This operation returns the ServiceInfo dataset for a service.

GetServiceCapabilities	0x9303	All object format and method format information is reported by using the GetServiceCapabilities operation.
GetServicePropDesc	0x9304	This operation returns theServicePropertyDesc dataset for a service.
GetServicePropList	0x9305	This operation is similar to GetObjectPropList in the MTP specification, Revision 1.0. GetServicePropList reads properties from a service.
SetServicePropList	0x9306	This operation sets a ServiceProperty by using the ServicePropList dataset. It enables the writing of property values to a service.
UpdateObjectPropList	0x9307	This operation sets the property list for a particular object that will be updated with a new binary object. This operation can be used to replace the binary data of an existing object.
DeleteObjectPropList	0x9308	This operation removes the properties that are specified in the DeleteObjectPropList dataset from the specified object or objects.
DeleteServicePropList	0x9309	This operation removes the properties that are specified in the DeleteServicePropList dataset from the specified service.

Design Notes:

- For information on defined MTP Services, refer to the MTP Device Services for Windows Specification available at http://msdn.microsoft.com/en-us/windows/hardware/gg463544.
- For information on service operations and general services details, refer to the MTP Device Services Extension Specification available at http://msdn.microsoft.com/en-us/windows/hardware/gg463545.
- See also Metadata Schema and Package Format Specification available at http://msdn.microsoft.com/en-us/windows/hardware/gg463153.

Device.Portable.Core.MediaSync

Portable Devices that support media content must meet basic interoperability requirements to successfully transfer content with an MTP aware media player application on Windows.

Applies to	Windows 10 x64 Windows 10 x86

Description

An MTP based device must be able to transfer data to and from a PC using a Windows based application that supports MTP.

Note that if the device replaces core MTP media handling functionality with a custom MTP Service, then interoperability with Windows Media Player will not function as expected. Devices that implement a custom service must have functional parity with core MTP operations to support media transfer and synchronization with a WPD-based application.

The following requirements must be met when interacting with Windows Media Player:

Supports digital media transfers from device to computer - The device must support digital media transfers to computers through applications that enable such transfers, such as the Portable Device Shell Namespace Extension and Windows Media Player.

Supports digital media transfers from computer to device - The device must support the transfer of supported media formats from the computer to the device through Windows Media Player. The device must support the transfer of all file formats from the computer to the device through the Windows® Namespace Extension.

Supports metadata updates - The device must support updates to MTP metadata after the original track has been copied to the device. The metadata updates are performed using Windows Media Player.

Supports cancellation of transfer and synchronization - The device must support cancellations of transfers and synchronizations mid-task without crashing or hanging of the device.

Properly identifies file types - The device must not rely on the file name extension of a file to discover the file type. The device can use the MTP object format or can parse the file contents to determine the file type.

Provide device object metadata - If a host sends an empty value for Album, Artist, or Title, the device must display meaningful text in place of the empty metadata and allow the user to find the content when searching on the associated field in the device UI. For example, a device might use Unknown Artist for the artist when it receives an empty value for artist metadata. If the device supports displaying genre metadata, then if a host sends an empty value for genre, the device must display meaningful text in place of the empty metadata and allow the user to find the content when searching on the associated field in the device UI.

Playlist transfer - The device must support the transfer of playlists manually created by the user using Windows Media Player. When a manual playlist is transferred using Windows Media Player, both the playlist and the content the playlist references must reside on the device.

To receive playlists from Windows Media Player, a device must support object references. If object references are supported, they must be supported on all media file formats. Support for object references is indicated by supporting the following commands:

- GetObjectReferences (0x9810)

- SetObjectReferences (0x9811)

The device must also support the AbstractAudioVideoPlaylist (0xBA05) format. A playlist is sent as a 0-byte object with a list of references to object handles for all objects contained in the playlist.

Device.Portable.Core.ModelID

Portable devices may support the optional ModelID property to uniquely identify logical device functions on a multi-function device.

Applies to	Windows 10 x64 Windows 10 x86

Description

Plug and Play includes support for a DevNode property called DEVPKEY_Device_ModelId. This key is used to store the ModelID value that some devices will store in a device-class-specific or manufacturer-specific manner on the device.

The ModelID spans logical device functions on a multi-function device through an internal structure in Windows 7 called the Display Object that aggregates logical devices in a single "piece of plastic" representation. The ModelID can be as specific or as generic as the manufacturer chooses. For example, ModelID may differ between product models, colors of an individual model, or even individual devices. To support ModelID, the device property code 0xD302 along with required properties must be supported.

Design Notes:

- Refer to the MTP Device Services Extension Specification that is included with the Windows Portable Device Enabling Kit, available at http://msdn.microsoft.com/en-us/windows/hardware/gg463545.

Device.Portable.Core.MTP

Portable devices must support a core set of MTP operations and devices properties as defined in the Media Transport Protocol revision 1.0 or later, along with the device properties and object formats for the specific device type.

Applies to	Windows 10 x64 Windows 10 x86

Description

Media Transfer Protocol (MTP) is an extension of the Picture Transfer Protocol (ISO 15740). Both specifications include optional commands and operations, but some of these are required for certification. There is a core set of operations and device properties that must be met by each device category in order to be compliant with Windows. Implementation details for MTP are defined in the

Media Transfer Protocol Specification, Revision 1.0 or later and will be used as the reference for certification of all portable devices as part of the Windows Hardware Certification Program.

Supports MTP drivers

A portable device must work with the inbox MTP drivers that ship with Windows. This means that they must install and work immediately without requiring the installation of additional drivers or software.

Supports MTP by default

Portable devices must ship with MTP enabled as the default connection mode. Devices can support Mass Storage Class (MSC) in addition to MTP.

- The device must not have a persistent, user accessible setting to enable or disable the MTP or MSC mode.
- All storage volumes on a device (both internal and external) must be accessible using the MTP protocol.

A device can have a safe mode that allows the user to boot to the MSC mode for device recovery scenarios. After the user disconnects a device connected in safe mode, the device must resume normal operation.

Device Information Dataset

The device must support the MTP DeviceInfo dataset. The following table describes field-specific requirements that must be implemented:

Dataset Field	Requirement	Notes
Standard Version	R	This identifies the PTP version this device can support in hundredths. For MTP devices, this shall contain the value 100 (representing 1.00).
MTP Vendor Extension ID	R	This identifies the MTP vendor-extension version in use by this device.
MTP Version	R	This identifies the version of the MTP standard this device supports. For MTP devices implemented under MTP 1.0, this shall contain the value 100 (representing 1.00). For MTP devices implemented under MTP 2.0, this shall contain the value 200.
MTP Extensions	I	This string is used to identify any extension sets applied to MTP.
Functional Mode	R	Modes allow the device to express different states with different capabilities. If the device supports only one

		mode, this field shall contain the value 0x00000000. See the MTP spec for details.
Operations Supported	R	This field identifies by datacode all operations that this device supports in the current functional mode.
Events Supported	I	This field identifies by datacode all events that this device can generate in the current functional mode.
Device Properties Supported	R	This field identifies by datacode all device properties that this device supports in the current functional mode.
Capture Formats	I	This field identifies by datacode the object format codes for each format that this device can generate independently (that is, without the content being placed on the device).
Playback Formats	I	This field identifies by datacode the object format codes for each format that this device can understand and parse if placed on the device. If the device can carry unidentified binary objects without understanding them, it shall indicate this by including the Undefined Object (0x3000) code in its Playback formats.
Manufacturer	R	The MTP specification identifies this as an optional string; it is required for Hardware Certification. This field contains a human-readable string identifying the manufacturer of this device.
Model	R	The MTP specification identifies this as an optional string; it is required for Hardware Certification. This field contains a human-readable string identifying the model of this device.
Device Version	R	The MTP specification identifies this as an optional string; it is required for Hardware Certification. This field contains a human-readable string identifying the software or firmware version of this device.
Serial Number	R	A serial number must be unique among all devices sharing identical Model and Device Version fields.

Where:

R = Required

I = If Implemented (Optional)

N/A = Not Applicable

Supports Control Requests

Control requests enable an MTP initiator to send out-of-band requests to the MTP responder. The device must support the following requests.

- Get Status
- Cancel Request
- Reset Device Request

Required Operations

This core set of MTP operations and device properties must be met by each portable device in order to be compliant with Windows. Implementation details for each operation and device property are defined in the Media Transfer Protocol Specification, Revision 1.0 or later. The highlighted operations represent the core set of MTP operations required for all portable devices.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
GetDeviceInfo	0x1001	R	R	R	R	R
OpenSession	0x1002	R	R	R	R	R
CloseSession	0x1003	R	R	R	R	R
GetStorageIDs	0x1004	R	R	R	R	R
GetStorageInfo	0x1005	R	R	R	R	R
GetNumObjects	0x1006	R	R	R	R	R
GetObjectHandles	0x1007	R	R	R	R	R
GetObjectInfo	0x1008	R	R	R	R	R
GetObject	0x1009	R	R	R	R	R
GetDevicePropDesc	0x1014	R	R	R	R	R
GetDevicePropValue	0x1015	R	R	R	R	R

DeleteObject	0x100B	R	R	I	I	I
SetDevicePropValue	0x100A	R	R	I	I	I
SendObjectInfo	0x100C	R	R	I	I	I
SendObject	0x100D	R	R	I	I	I
GetPartialObject	0x101B	R	R	I	I	I
GetObjectPropsSupported	0x9801	R	R	I	I	I
GetObjectPropDesc	0x9802	R	R	I	I	I
GetObjectPropValue	0x9803	R	R	I	I	I
SetObjectPropValue	0x9804	R	R	I	I	I
GetObjectReferences	0x9810	R	R	I	I	I
SetObjectReferences	0x9811	R	R	I	I	I

See **Operations** in the MTP Specification, Revision 1.0 or later for complete list of defined MTP Operations.

The following operations are highly recommended, though not required:

- FormatStore (0x100F)
- GetObjectPropList (0x9805)
- SetObjectPropList (0x9806)
- GetInterDependentPropDesc (0x9807)
- SendObjectPropList (0x9808) - For this command, your device must also support specification of destination, which allows the MTP initiator to dictate a parent handle for that object.

Required Events

The following events must be supported for all objects:

- ObjectAdded (0x4002)
- ObjectRemoved (0x4003)

If the device supports removable media, it must also support the following events.

- StoreAdded (0x4004)
- StoreRemoved (0x4005)

Operation Responses

An appropriate response must be returned for any and all operations. If an error is encountered, error response codes are also defined and must be provided by the device. For more information, see the "Responses" section of the MTP specification.

Required Device Properties

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Battery Level	0x5001	I	I	I	I	I
Synchronization Partner	0xD401	I	I	I	I	I
Device Friendly Name	0xD402	R	R	I	I	I
Device Icon	0xD405	I	I	I	I	I

See **Device Properties** in the MTP Specification, Revision 1.0 or later for a complete list of defined Device Properties.

Device Icon is not required, but is strongly recommended. High-resolution images of the device can be used and exposed in the Windows UI.

Object Formats

The following table represents the list of required object formats for a portable device. Each object format also has a list of object properties that must be supported per object type. The highlighted object format is required for all MTP devices and is considered a core MTP requirement for Windows.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Undefined	0x3000	I	I	I	I	I
Association	0x3001	R	R	R	R	R

Abstract Audio Album	0xBA03	I(2)	I(2)	N/A	N/A	I(2)
Abstract Audio & Video Playlist	0xBA05	I	I	N/A	N/A	I
WAV	0x3008	R(1)	R(1)	I	I	I
MP3	0x3009	R(1)	R(1)	I	I	I
AVI	0x300A	R(1)	R(1)	I	R(1)	I
MPEG	0x300B	R(1)	R(1)	I	R(1)	I
ASF	0x300C	R(1)	R(1)	I	R(1)	I
EXIF/JPEG	0x3801	I	I	R(1)	I	I
TIFF/EP	0x3802	I	I	R(1)	I	I
BMP	0x3804	I	I	R(1)	I	I
JPEG XR	0xB804	I	I	R(1)	I	I
WMA	0xB901	R(1)	R(1)	I	I	I
AAC	0xB903	R(1)	R(1)	I	I	I
WMV	0xB981	I	I	I	R(1)	I
MP4 Container	0xB982	I	I	I	R(1)	I
3GP Container	0xB984	I	I	I	R(1)	I
3G2	0xB985	I	I	I	R(1)	I
AVCHD	0xB986	I	I	I	R(1)	I

(1) Portable devices that are capable of playing audio and/or video must support at least one of these formats. This table does not represent the complete list of supported formats; it represents the list of commonly used formats.

See **Object Formats** in the MTP Specification, Revision 1.0 or later for complete list of defined Device Properties.

(2) Support for the following object properties is mandatory for AbstractAudioAlbum objects:

- Genre (0xDC8C)
- Album Artist (0xDC9B)
- WMPMetadataRoundTrip (0x9201)

A device that supports both the AbstractAudioAlbum format and an image format may optionally support album art. Album art is attached to an AbstractAudioAlbum object by adding the art to the album's RepresentativeSampleData property.

If a Portable Device supports album art, then the device must support the following object properties:

- PurchaseFlag (0xd901), an object property in the Windows Media DRM 10 for Portable Devices MTP Extensions
- RepresentativeSampleFormat (0xdc81)
- RepresentativeSampleSize (0xdc82)
- RepresentativeSampleHeight (0xdc83)
- RepresentativeSampleWidth (0xdc84)
- RepresentativeSampleData (0xdc86)
- User Rating (0xdc8a)
- WMPMetadataRoundTrip (0x9201)

Design Notes:

- "Picture Transfer Protocol (PTP) for Digital Still Photography Devices," Version 1.0 of the PIMA15740: 2000 Picture Transfer Protocol specification.
- The Media Transfer Protocol Specification, Revision 1.0 is available at http://go.microsoft.com/fwlink/?LinkId=243145.
- Additional implementation details can be found in the Windows 7 Portable Device Enabling Kit for MTP, which is available at http://go.microsoft.com/fwlink/?LinkId=243146.

Device.Portable.Core.MTPFunctionality

A device that supports MTP must meet mandatory general functionality requirements to ensure expected behavior in Windows.

Applies to	Windows 10 x64 Windows 10 x86

Description

Portable devices must behave according to the requirements defined below. The device must:

Persist all transferred content - The device must not report receiving content that it has not persisted.

Respond as expected - A portable device must respond when it receives an MTP GetDeviceStatus control request issued by a host.

Support unexpected device disconnects - An unexpected disconnect must not cause the device to stop responding (hang or crash) or to restart.

Device.Portable.Core.MTPMultiSession

Portable devices that enable MTP multisession functionality must support required object session operations.

Applies to	Windows 10 x64 Windows 10 x86

Description

In order to properly support MTP multisession functionality and work as expected with a multisession aware Initiator, the device must support the following session operations:

- CreateSession
- GetSessionResponderInfo

The RestrictSession operation is optional. This operation is supported in Windows and will honor restrictions that are defined in a responders RestrictSession dataset.

These operations are required in addition to the operations for basic session operation defined in the MTP 1.0 specification; those operations include OpenSession, CloseSession, TransactionID, SessionID, etc. There is no requirement for the minimum or maximum number of sessions that a device must support if multisession.

Design Notes:

- For implementation details see the Media Transfer Protocol Specification Revision 2.0, available at http://go.microsoft.com/fwlink/?LinkId=243142.

Device.Portable.Core.MTPObjectProperties

A MTP device must support object properties for each consumable media format.

Applies to	Windows 10 x64 Windows 10 x86

Description

The following device capabilities must be supported for each format reported by the device. If an MP4 container is supported, the device capability needs to include the proper properties in this entry. Based on this information, Windows can predict the proper transcode profile and transcode a file to the device that can then be played back on the device.

The following table summarizes Required (R) and recommended If Implemented (I) / Optional object property support for all object formats. This is not an exhaustive list of object properties available, for the complete list refer to the Object Format Summary Table in the MTP specification.

Object property	MTP Datacode	Status	**Description**
Storage ID	0xDC01	R	The storage area that the object is stored in.
Object Format	0xDC02	R	The data format for the object.
Protection Status	0xDC03	R	The protection status of the object.
Object Size	0xDC04	R	The size of the data component of the object, in bytes.
Object File Name	0xDC07	R	The file name of the object.
Date Created	0xDC08	I**(1)**	This property contains the date and time when the object was created.
Date Modified	0xDC09	I**(1)**	The date and time when the object was last altered.
Parent Object	0xDC0B	R	The parent object of the object.

Persistent Unique Object Identifier	0xDC41	R	The persistent unique object identifier of the object.
Name	0xDC44	R	The name of the object.
Non-Consumable	0xDC4F	R	Indicates whether the object was transferred to the portable media player for storage only and is, therefore, not available to be consumed (for example, played) by the portable device.

Where:

R = Required

I = If Implemented (Optional)

N/A = Not Applicable

(1) It is strongly recommended that Date Created and Date Modified be supported in order for applications to be able to distinguish which objects have been previously imported or synchronized. This is particularly important for photo acquisition scenarios.

The following table summarizes required and If Implemented (Optional) object property support for image, audio, and video objects. This is not an exhaustive list of all supported object properties. For a complete list, refer to the Object Property Summary Table in the MTP Specification.

Object property	MTP Datacode	Image	Audio	Video
Artist	0xDC46	N/A	R	I
Description	0xDC48	N/A	I	I
Representative Sample Format	0xDC81	I	I	I
Representative Sample Size	0xDC82	I	I	I
Representative Sample Height	0xDC83	I	I	I
Representative Sample Width	0xDC84	I	I	I
Representative Sample Data	0xDC86	I	I	I
Width	0xDC87	R	N/A	R

Height	0xDC88	R	N/A	R
Duration	0xDC89	N/A	I	I
User Rating	0xDC8a	N/A	I	I
Track	0xDC8b	N/A	R	I
Genre	0xDC8c	N/A	I	I
Use Count	0xDC91	N/A	I	I
Parental Rating	0xDC94	N/A	I	I
Original Release Date	0xDC99	N/A	I	I
Album Name	0xDC9A	N/A	R	N/A
Album Artist	0xDC9B	N/A	R**(2)**	N/A
Bitrate Type	0xDE92	N/A	I	I
Sample Rate	0xDE93	N/A	R	R
Number of Channels	0xDE94	N/A	R	R
ScanType	0xDE97	N/A	I	R
Audio WAVE Codec	0xDE99	N/A	R	R
Audio Bitrate	0xDE9A	N/A	R	R
Video FourCC Codec	0xDE9B	N/A	N/A	R
Video Bitrate	0xDE9C	N/A	N/A	R
Frames Per Thousand Seconds	0xDE9D	N/A	N/A	R
Key Frame Distance	0xDE9E	N/A	N/A	I

Encoding Profile	0xDEA1	N/A	I	R

(2) Note that Album Artist (0xDC9B) is only required if the entire album is available, individual audio files only need to support artist (0xDC46).

Additionally, if the device supports an If Implemented (Optional) object property, the device must do so in conformance with the MTP specification and guidelines. Tests are designed to validate optional functionality if the device supports it.

Design Notes:

For implementation details, refer to Media Transfer Protocol Specification Revision 1.0, Appendix B - Object Properties available at http://go.microsoft.com/fwlink/?LinkId=243143.

Device.Portable.Core.MTPStreams

Portable devices that implement MTP Streams must support required object stream operations.

Applies to	Windows 10 x64 Windows 10 x86

Description

In order to properly support MTP Stream functionality, a device must support the following operations:

Operation	Status	**Description**
OpenObjectStream	R	This operation opens an object as a stream.
ReadObjectStream	R	This operation reads a portion of data from the previously opened object stream.
WriteObjectStream	R	This operation writes the portion of data to the previously opened object stream.
SeekObjectStream	R	This operation seeks forward reading or writing of the next data block in the stream.
CloseObjectStream	R	This operation closes the previously opened object stream and releases the allocated

Where:

R = Required

I = If Implemented (Optional)

N/A = Not Applicable

Design Notes:

- For implementation details refer to Media Transfer Protocol Specification Revision 2.0, available at http://go.microsoft.com/fwlink/?LinkId=243144.

Device.Portable.Core.TransportBluetooth

If a portable device leverages the Bluetooth transport, then the device shall support the latest required specification(s) for that transport and related tests: Bluetooth (Specification Version 2.1 or greater).

Applies to	Windows 10 x64 Windows 10 x86

Description

A portable device that uses Bluetooth must use the following specifications:

- Bluetooth Core specification Version 2.1 + EDR (with latest errata) or newer [https://www.bluetooth.org/Technical/Specifications/adopted.htm]
- MTP over Bluetooth Profile Specification from Microsoft [http://msdn.microsoft.com/en-us/windows/hardware/gg463543]

Design Notes:

- The connection must be implemented according to requirements defined for Bluetooth devices.
- A Bluetooth-capable portable device must be able to communicate with Window's Bluetooth Class Driver (in a standard MTP conversation over Bluetooth similar to USB and TCP/IP).
- A Bluetooth-capable portable device must support Bluetooth V2.1+EDR, to ensure that Windows can leverage Secure Simple Pairing (SSR) for optimized pairing experience.
- A Bluetooth-capable portable device must leverage L2CAP Transport MTP Responder Service Definition Record required parameter "GetFormatCapabilities" to mitigate format enumeration performance issues.

Device.Portable.Core.TransportIP

If a portable device leverages the IP transport, then the device shall support the latest required specification(s) for that transport and related tests: IP (Along with the MTP Network Association Extension specification and related UPnP specifications).

Applies to	Windows 10 x64 Windows 10 x86

Description

A Windows portable device that supports MTP over IP must comply with the Camera and Imaging Products Association (CIPA) PTP over Internet Protocol (PTP/IP) specification (CIPA-005/2005), which extends the PTP specification to use TCP/IP to perform PTP operations over wireless networks.

An MTP/IP device must support the following three technologies to ensure compatibility with the PC:

1. UPnP Basic Device

MTP/IP devices must support Simple Service Discovery Protocol (SSDP) discovery as a Universal Plug and Play (UPnP„¢) basic device. Refer to the MTP Network Association Extension definition for details.

2. Picture Transfer Protocol over IP

MTP/IP devices must comply with the Camera and Imaging Products Association (CIPA) PTP over Internet Protocol (PTP/IP) specification (CIPA-005/2005), which extends the PTP specification to use TCP/IP to perform PTP operations over wireless networks.

3. MTP Network Association Extension

MTP/IP devices must comply with the Wi-Fi-provisioning extension to the MTP specification and the MTP Network Association extension to the MTP specification.

Support for this MTP extension is indicated by including the following string in the MTPVendorExtensionDesc field of the MTP DeviceInfo dataset, returned as a response to the MTP GetDeviceInfo operation.

Dataset field	Datatype
MTPVendorExtensionDesc	"microsoft.com/WPDNA: 1.0"

The device property defined by this extension supports Network Association configuration for devices that support Zero or Nominal Authentication. Secure Authentication is not supported by this extension; secure authentication can be implemented via a custom MTP Service if desired. Refer to the MTP Network Association Extension definition for details.

Recommendations for Network Provisioning

The MTP Wi-Fi Provisioning Extension extends the Windows Connect Now architecture to MTP devices. This extension defines a new MTP object format for WFC (Wireless Configuration File) objects. A WFC object contains the network settings that will allow the responder to join a wireless network. An initiator supporting the WCN architecture will be able to transfer WFC objects to the device. Each WFC object represents the settings for a single wireless network. The responder may receive multiple WFC objects over time. Each object will be named according to the SSID of the network.

The following DataType must be included in the MTPVendorExtensionDesc field of the DeviceInfo dataset.

Dataset field	Datatype
MTPVendorExtensionDesc	"microsoft.com/WPDWCN: 1.0"

Recommendations for Improved Performance over IP

In order to mitigate format enumeration performance issues, MTP/IP devices should also support the **GetFormatCapabilities** operation as defined in the MTP Device Services Extensions Specification. This operation improves the performance of querying a device for the supported object properties on formats that are associated with classic MTP storages (those formats that appear in the DeviceInfo dataset). GetFormatCapabilities is a bulk operation that duplicates the functionality of GetObjectPropsSupported, GetObjectPropDesc, and GetInterdependentPropDesc. Responders should implement this operation because it replaces multiple calls to the device with a single, more efficient call.

Design Notes:

- Refer to the PTP/IP Specification (CIPA-005/2005) "Picture Transfer Protocol over TCP/IP Networks".
- MTP Device Services Extension Specification at http://msdn.microsoft.com/en-us/windows/hardware/gg463545.
- MTP Network Association, MTP Windows Connect Now, and MTP Wi-Fi Provisioning Extension documents are available at http://msdn.microsoft.com/en-us/windows/hardware/gg463543.

Device.Portable.Core.TransportIPDLNA

A Portable Device that functions as a DMR or DMS conforms to requirements defined for Networked Media Devices.

Applies to	Windows 10 x64 Windows 10 x86

Description

Portable devices can implement one or more of the DLNA device classes such as:

- Digital Media Renderer (DMR)
- Digital Media Server (DMS)

The device must meet the requirements defined for each device class called out in the Networked Media Device section of the Windows Hardware Certification Program Requirements. DLNA is managed independent of MTP.

Design Notes:

- Refer to the Networked Media Device section for requirement details. Information on DLNA Certification can be found at http://www.dlna.org.

Device.Portable.Core.TransportUSB

If a Portable Device leverages the USB transport, then the device shall support the latest required specification(s) for that transport and related tests: USB (Specification Version 2.0 or greater).

Applies to	Windows 10 x64 Windows 10 x86

Description

A portable device that uses USB must use the following:

- USB Core specification Version 2.0 (with latest errata) or newer [http://www.usb.org/developers/docs]
- USB MTP DWG Specification 1.0 (with latest errata) or newer [http://www.usb.org/developers/devclass_docs]

Design Notes:

- USB connected devices must support both High-Speed and Full-Speed. Super-Speed is optional.
- The connection must be implemented according to requirements in the appropriate connection type section.
- There are no specific Windows Hardware Certification requirements for the type of USB connector used on a portable device.

Device.Portable.Core.VideoCodec

If a portable device has the ability to capture video content, it must do so using a format supported natively in Windows.

Applies to	Windows 10 x64 Windows 10 x86

Description

If a portable device can capture video content, it must do so in a format that Windows can decode natively without the need for additional codecs to be installed.

The following tables represent the list of in-box formats that Windows will render to; this is not the exhaustive of supported formats. The full list of formats supported can be found at: http://go.microsoft.com/fwlink/p/?linkid=242995&clcid=0x409.

Note that for a given format in the tables below it is not required to support all given Bit rates and Sample rates, it is however required to support a format that lines up within the ranges provided.

MP4 Container Content

	Setting	Requirement
AAC	Codec	AAC Standard, minimum AAC Profile level 2 (0x29), compatible with 0x29, 0x2A, 0x2B, 0x2C, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33
	bit rate for CBR	96, 128, 160, 192 Kbps
	Sample rate	44.1 or 48 kHz
	Channels	2
H.264	Codec	H.264 Standard
	Codec profile	Baseline, Level 3
	Resolution, pixel aspect ratio	Any (as long as reported)
	Frame rate	Any (as long as reported)
	Average bit rate	Any (as long as reported)

Mobile WMV Video Content

	Setting	Requirement
WMA Standard	Codec	WMA9 or later
	Average bit rate	From 48 to 96 kilobits per second (Kbps)
	Maximum peak bit rate	192 Kbps
	Sample rate	44.1 or 48 kHz
	Channels	2
WMV	Codec	VC-1
	Codec profile	Simple
	Resolution, pixel aspect ratio	One of the following: · 176Ã—144 pixels, 1:1 · 220Ã—176 pixels, 1:1
	Frame rate	15 or 24 frames per second
	Average bit rate	From 96 to 384 Kbps
	Maximum peak bit rate	768 Kbps
	Maximum buffer	3 seconds
	Maximum key-frame distance	15 seconds
	Color depth	16 bits per pixel

Portable WMV Video Content

	Setting	Requirement

WMA Standard	Codec	WMA9 or later
	Average bit rate	From 48 to 128 Kbps
	Maximum peak bit rate	256Kbps
	Sample rate	44.1 or 48 kHz
	Channels	2
WMV	Codec	VC-1
	Codec profile	Simple
	Resolution, pixel aspect ratio	One of the following: · 320 Ã—240 pixels, 1:1 · 480 Ã—270 pixels, 1:1
	Frame rate	24, 25, or 29.97 frames per second
	Average bit rate	From 384 to 850 Kbps
	Maximum peak bit rate	1,700 Kbps
	Maximum buffer	3 seconds
	Maximum key-frame distance	15 seconds
	Color depth	16 bits per pixel

Standard WMV Video Content

	Setting	Requirement
WMA Standard	Codec	WMA9 or later
	Average bit rate	From 64 to 192 Kbps

	Maximum peak bit rate	360 Kbps
	Sample rate	44.1 or 48 kHz
	Channels	2
WMV	Codec	VC-1
	Codec profile	Main
	Resolution, pixel aspect ratio	One of the following: · 640 Ã—480 pixels, 1:1 · 720 Ã—480 pixels, 10:11 · 720 Ã—480 pixels, 40:33
	Frame rate	24, 25, or 29.97 frames per second
	Average bit rate	From 1,000 to 3,000 Kbps
	Maximum peak bit rate	6,000 Kbps
	Maximum buffer	2 seconds
	Maximum key-frame distance	15 seconds
	Color depth	16 or 24 bits per pixel

High Definition WMV Video Content

	Setting	Minimum requirement
WMA Standard	Codec	WMA9 or later
	Average bit rate	From 64 to 192 Kbps
	Maximum peak bit rate	360 Kbps
	Sample rate	44.1 or 48 kHz

	Channels	2
WMA Professional	Codec	WMA9 or later
	Average bit rate	From 320 to 1,500 Kbps
	Maximum peak bit rate	2,500 Kbps
	Sample rate	44.1, 48, or 96 kHz
	Channels	up to 8
WMV	Codec	VC-1
	Codec profile	Advanced
	Resolution, pixel aspect ratio	1280 Ã—720 pixels, 1:1
	Frame rate	24, 25, or 29.97 frames per second
	Average bit rate	From 8,000 to 10,000 Kbps
	Maximum peak bit rate	20,000 Kbps
	Maximum buffer	2 seconds
	Maximum key-frame distance	15 seconds
	Color depth	24 bits per pixel

Send comments about this topic to Microsoft

Device.Portable.DigitalCamera

DigitalCamera

In this topic:

Device.Portable.DigitalCamera.MTP

Digital Cameras must support MTP operations and properties as defined in the Media Transport Protocol revision 1.0 or later, along with specific object formats per device type.

Applies to	Windows 10 x64 Windows 10 x86

Description

Media Transfer Protocol (MTP) is an extension of the Picture Transfer Protocol (ISO 15740). Both specifications include optional commands and operations, but some of these are required for certification. There is a core set of operations and device properties that must be met by each device category in order to be compliant with Windows. Implementation details for MTP are defined in the Media Transfer Protocol Specification, Revision 1.0 or later and will be used as the reference for certification of all portable devices as part of the Windows Hardware Certification Program.

Required Operations

This core set of MTP operations and device properties must be met by each portable device in order to be compliant with Windows. Implementation details for each operation and device property are defined in the Media Transfer Protocol Specification, Revision 1.0 or later.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
GetDeviceInfo	0x1001	R	R	R	R	R
OpenSession	0x1002	R	R	R	R	R
CloseSession	0x1003	R	R	R	R	R
GetStorageIDs	0x1004	R	R	R	R	R
GetStorageInfo	0x1005	R	R	R	R	R
GetNumObjects	0x1006	R	R	R	R	R
GetObjectHandles	0x1007	R	R	R	R	R

GetObjectInfo	0x1008	R	R	R	R	R
GetObject	0x1009	R	R	R	R	R
GetDevicePropDesc	0x1014	R	R	R	R	R
GetDevicePropValue	0x1015	R	R	R	R	R
DeleteObject	0x100B	R	R	I	I	I
SetDevicePropValue	0x100A	R	R	I	I	I
SendObjectInfo	0x100C	R	R	I	I	I
SendObject	0x100D	R	R	I	I	I
GetPartialObject	0x101B	R	R	I	I	I
GetObjectPropsSupported	0x9801	R	R	I	I	I
GetObjectPropDesc	0x9802	R	R	I	I	I
GetObjectPropValue	0x9803	R	R	I	I	I
SetObjectPropValue	0x9804	R	R	I	I	I
GetObjectReferences	0x9810	R	R	I	I	I
SetObjectReferences	0x9811	R	R	I	I	I

Where:

R = Required

I = If Implemented (Optional)

N/A = Not Applicable

See **Operations** in the MTP Specification, Revision 1.0 or later for complete list of defined MTP Operations.

The following operations are highly recommended, though not required:

- FormatStore (0x100F)
- GetObjectPropList (0x9805)
- SetObjectPropList (0x9806)
- GetInterDependentPropDesc (0x9807)
- SendObjectPropList (0x9808) - For this command, your device must also support specification of destination, which allows the MTP initiator to dictate a parent handle for that object.

Required Events

The following events must be supported for all objects:

- ObjectAdded (0x4002)
- ObjectRemoved (0x4003)

If the device supports removable media, it must also support the following events.

- StoreAdded (0x4004)
- StoreRemoved (0x4005)

Operation Responses

An appropriate response must be returned for any and all operations. If an error is encountered, error response codes are also defined and must be provided by the device. For more information, see the "Responses" section of the MTP specification.

Required Device Properties

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Battery Level	0x5001	I	I	I	I	I
Synchronization Partner	0xD401	I	I	I	I	I
Device Friendly Name	0xD402	R	R	I	I	I
Device Icon	0xD405	I	I	I	I	I

See **Device Properties** in the MTP Specification, Revision 1.0 or later for a complete list of defined Device Properties.

Device Icon is not required but is strongly recommended. High-resolution images of the device can be used and exposed in the Windows UI.

Object Formats

The following table represents the list of required object formats for a portable device.

Each object format also has a list of object properties that must be supported per object type.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Undefined	0x3000	I	I	I	I	I
Association	0x3001	R	R	R	R	R
Abstract Audio Album	0xBA03	I	I	N/A	N/A	I
Abstract Audio & Video Playlist	0xBA05	I	I	N/A	N/A	I
WAV	0x3008	R(1)	R(1)	I	I	I
MP3	0x3009	R(1)	R(1)	I	I	I
AVI	0x300A	R(1)	R(1)	I	R(1)	I
MPEG	0x300B	R(1)	R(1)	I	R(1)	I
ASF	0x300C	R(1)	R(1)	I	R(1)	I
EXIF/JPEG	0x3801	I	I	R(1)	I	I
TIFF/EP	0x3802	I	I	R(1)	I	I
BMP	0x3804	I	I	R(1)	I	I
JPEG XR	0xB804	I	I	R(1)	I	I
WMA	0xB901	R(1)	R(1)	I	I	I
AAC	0xB903	R(1)	R(1)	I	I	I

WMV	0xB981	I	I	I	R(1)	I
MP4 Container	0xB982	I	I	I	R(1)	I
3GP Container	0xB984	I	I	I	R(1)	I
3G2	0xB985	I	I	I	R(1)	I
AVCHD	0xB986	I	I	I	R(1)	I

(1) Portable devices that are capable of playing audio and/or video must support at least one of these formats. This table does not represent the complete list of supported formats; it represents the list of commonly used formats.

See **Object Formats** in the MTP Specification, Revision 1.0 or later for complete list of defined Device Properties.

Design Notes:

- "Picture Transfer Protocol (PTP) for Digital Still Photography Devices," Version 1.0 of the PIMA15740: 2000 Picture Transfer Protocol specification.
- The Media Transfer Protocol Specification, Revision 1.0 is available at http://go.microsoft.com/fwlink/?LinkId=243145.
- Additional implementation details can be found in the Windows 7 Portable Device Enabling Kit for MTP, which is available at http://go.microsoft.com/fwlink/?LinkId=243146.

Send comments about this topic to Microsoft

Device.Portable.DigitalVideoCamera

DigitalVideoCamera

In this topic:

Device.Portable.DigitalVideoCamera.MTP

Digital video cameras must support MTP operations and properties as defined in the Media Transport Protocol revision 1.0 or later, along with specific object formats per device type.

Applies to	Windows 10 x64 Windows 10 x86

Description

Media Transfer Protocol (MTP) is an extension of the Picture Transfer Protocol (ISO 15740). Both specifications include optional commands and operations, but some of these are required for certification. There is a core set of operations and device properties that must be met by each device category in order to be compliant with Windows. Implementation details for MTP are defined in the Media Transfer Protocol Specification, Revision 1.0 or later and will be used as the reference for certification of all portable devices as part of the Windows Hardware Certification Program.

Required Operations

This core set of MTP operations and device properties must be met by each portable device in order to be compliant with Windows. Implementation details for each operation and device property are defined in the Media Transfer Protocol Specification, Revision 1.0 or later.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
GetDeviceInfo	0x1001	R	R	R	R	R
OpenSession	0x1002	R	R	R	R	R
CloseSession	0x1003	R	R	R	R	R
GetStorageIDs	0x1004	R	R	R	R	R
GetStorageInfo	0x1005	R	R	R	R	R
GetNumObjects	0x1006	R	R	R	R	R
GetObjectHandles	0x1007	R	R	R	R	R
GetObjectInfo	0x1008	R	R	R	R	R
GetObject	0x1009	R	R	R	R	R
GetDevicePropDesc	0x1014	R	R	R	R	R
GetDevicePropValue	0x1015	R	R	R	R	R

DeleteObject	0x100B	R	R	I	I	I
SetDevicePropValue	0x100A	R	R	I	I	I
SendObjectInfo	0x100C	R	R	I	I	I
SendObject	0x100D	R	R	I	I	I
GetPartialObject	0x101B	R	R	I	I	I
GetObjectPropsSupported	0x9801	R	R	I	I	I
GetObjectPropDesc	0x9802	R	R	I	I	I
GetObjectPropValue	0x9803	R	R	I	I	I
SetObjectPropValue	0x9804	R	R	I	I	I
GetObjectReferences	0x9810	R	R	I	I	I
SetObjectReferences	0x9811	R	R	I	I	I

Where:

R = Required

I = If Implemented (Optional)

N/A = Not Applicable

See **Operations** in the MTP Specification, Revision 1.0 or later for complete list of defined MTP Operations.

The following operations are highly recommended, though not required:

- FormatStore (0x100F)
- GetObjectPropList (0x9805)
- SetObjectPropList (0x9806)
- GetInterDependentPropDesc (0x9807)

- SendObjectPropList (0x9808) - For this command, your device must also support specification of destination, which allows the MTP initiator to dictate a parent handle for that object.

Required Events

The following events must be supported for all objects:

- ObjectAdded (0x4002)
- ObjectRemoved (0x4003)

If the device supports removable media, it must also support the following events.

- StoreAdded (0x4004)
- StoreRemoved (0x4005)

Operation Responses

An appropriate response must be returned for any and all operations. If an error is encountered, error response codes are also defined and must be provided by the device. For more information, see the "Responses" section of the MTP specification.

Required Device Properties

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Battery Level	0x5001	I	I	I	I	I
Synchronization Partner	0xD401	I	I	I	I	I
Device Friendly Name	0xD402	R	R	I	I	I
Device Icon	0xD405	I	I	I	I	I

See **Device Properties** in the MTP Specification, Revision 1.0 or later for a complete list of defined Device Properties.

Device Icon is not required but is strongly recommended. High-resolution images of the device can be used and exposed in the Windows UI.

Object Formats

The following table represents the list of required object formats for a portable device.

Each object format also has a list of object properties that must be supported per object type.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Undefined	0x3000	I	I	I	I	I
Association	0x3001	R	R	R	R	R
Abstract Audio Album	0xBA03	I	I	N/A	N/A	I
Abstract Audio & Video Playlist	0xBA05	I	I	N/A	N/A	I
WAV	0x3008	R(1)	R(1)	I	I	I
MP3	0x3009	R(1)	R(1)	I	I	I
AVI	0x300A	R(1)	R(1)	I	R(1)	I
MPEG	0x300B	R(1)	R(1)	I	R(1)	I
ASF	0x300C	R(1)	R(1)	I	R(1)	I
EXIF/JPEG	0x3801	I	I	R(1)	I	I
TIFF/EP	0x3802	I	I	R(1)	I	I
BMP	0x3804	I	I	R(1)	I	I
JPEG XR	0xB804	I	I	R(1)	I	I
WMA	0xB901	R(1)	R(1)	I	I	I
AAC	0xB903	R(1)	R(1)	I	I	I
WMV	0xB981	I	I	I	R(1)	I
MP4 Container	0xB982	I	I	I	R(1)	I
3GP Container	0xB984	I	I	I	R(1)	I

3G2	0xB985	I	I	I	R(1)	I
AVCHD	0xB986	I	I	I	R(1)	I

(1) Portable devices that are capable of playing audio and/or video must support at least one of these formats. This table does not represent the complete list of supported formats; it represents the list of commonly used formats.

See **Object Formats** in the MTP Specification, Revision 1.0 or later for complete list of defined Device Properties.

Design Notes:

- "Picture Transfer Protocol (PTP) for Digital Still Photography Devices," Version 1.0 of the PIMA15740: 2000 Picture Transfer Protocol specification.
- The Media Transfer Protocol Specification, Revision 1.0 is available at http://go.microsoft.com/fwlink/?LinkId=243145.
- Additional implementation details can be found in the Windows 7 Portable Device Enabling Kit for MTP, which is available at http://go.microsoft.com/fwlink/?LinkId=243146.

Send comments about this topic to Microsoft

Device.Portable.MediaPlayer

MediaPlayer

In this topic:

Device.Portable.MediaPlayer.MTP

Media players must support MTP operations and properties as defined in the Media Transport Protocol revision 1.0 or later, along with specific object formats per device type.

Applies to	Windows 10 x64 Windows 10 x86

Description

Media Transfer Protocol (MTP) is an extension of the Picture Transfer Protocol (ISO 15740). Both specifications include optional commands and operations, but some of these are required for certification. There is a core set of operations and device properties that must be met by each device category in order to be compliant with Windows. Implementation details for MTP are defined in the Media Transfer Protocol Specification, Revision 1.0 or later and will be used as the reference for certification of all portable devices as part of the Windows Hardware Certification Program.

Required Operations

This core set of MTP operations and device properties must be met by each portable device in order to be compliant with Windows. Implementation details for each operation and device property are defined in the Media Transfer Protocol Specification, Revision 1.0 or later.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
GetDeviceInfo	0x1001	R	R	R	R	R
OpenSession	0x1002	R	R	R	R	R
CloseSession	0x1003	R	R	R	R	R
GetStorageIDs	0x1004	R	R	R	R	R
GetStorageInfo	0x1005	R	R	R	R	R
GetNumObjects	0x1006	R	R	R	R	R
GetObjectHandles	0x1007	R	R	R	R	R
GetObjectInfo	0x1008	R	R	R	R	R
GetObject	0x1009	R	R	R	R	R
GetDevicePropDesc	0x1014	R	R	R	R	R
GetDevicePropValue	0x1015	R	R	R	R	R
DeleteObject	0x100B	R	R	I	I	I
SetDevicePropValue	0x100A	R	R	I	I	I
SendObjectInfo	0x100C	R	R	I	I	I

SendObject	0x100D	R	R	I	I	I
GetPartialObject	0x101B	R	R	I	I	I
GetObjectPropsSupported	0x9801	R	R	I	I	I
GetObjectPropDesc	0x9802	R	R	I	I	I
GetObjectPropValue	0x9803	R	R	I	I	I
SetObjectPropValue	0x9804	R	R	I	I	I
GetObjectReferences	0x9810	R	R	I	I	I
SetObjectReferences	0x9811	R	R	I	I	I

Where:

R = Required

I = If Implemented (Optional)

N/A = Not Applicable

See **Operations** in the MTP Specification, Revision 1.0 or later for complete list of defined MTP Operations.

The following operations are highly recommended, though not required:

- FormatStore (0x100F)
- GetObjectPropList (0x9805)
- SetObjectPropList (0x9806)
- GetInterDependentPropDesc (0x9807)
- SendObjectPropList (0x9808) - For this command, your device must also support specification of destination, which allows the MTP initiator to dictate a parent handle for that object.

Required Events

The following events must be supported for all objects:

- ObjectAdded (0x4002)
- ObjectRemoved (0x4003)

If the device supports removable media, it must also support the following events.

- StoreAdded (0x4004)
- StoreRemoved (0x4005)

Operation Responses

An appropriate response must be returned for any and all operations. If an error is encountered, error response codes are also defined and must be provided by the device. For more information, see the "Responses" section of the MTP specification.

Required Device Properties

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Battery Level	0x5001	I	I	I	I	I
Synchronization Partner	0xD401	I	I	I	I	I
Device Friendly Name	0xD402	R	R	I	I	I
Device Icon	0xD405	I	I	I	I	I

See **Device Properties** in the MTP Specification, Revision 1.0 or later for complete list of defined Device Properties.

Device Icon is not required but is strongly recommended. High-resolution images of the device can be used and exposed in the Windows UI.

Object Formats

The following table represents the list of required object formats for a portable device.

Each object format also has a list of object properties that must be supported per object type.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Undefined	0x3000	I	I	I	I	I
Association	0x3001	R	R	R	R	R

Abstract Audio Album	0xBA03	I(2)	I(2)	N/A	N/A	I(2)
Abstract Audio & Video Playlist	0xBA05	I	I	N/A	N/A	I
WAV	0x3008	R(1)	R(1)	I	I	I
MP3	0x3009	R(1)	R(1)	I	I	I
AVI	0x300A	R(1)	R(1)	I	R(1)	I
MPEG	0x300B	R(1)	R(1)	I	R(1)	I
ASF	0x300C	R(1)	R(1)	I	R(1)	I
EXIF/JPEG	0x3801	I	I	R(1)	I	I
TIFF/EP	0x3802	I	I	R(1)	I	I
BMP	0x3804	I	I	R(1)	I	I
JPEG XR	0xB804	I	I	R(1)	I	I
WMA	0xB901	R(1)	R(1)	I	I	I
AAC	0xB903	R(1)	R(1)	I	I	I
WMV	0xB981	I	I	I	R(1)	I
MP4 Container	0xB982	I	I	I	R(1)	I
3GP Container	0xB984	I	I	I	R(1)	I
3G2	0xB985	I	I	I	R(1)	I
AVCHD	0xB986	I	I	I	R(1)	I

(1) Portable devices that are capable of playing audio and/or video must support at least one of these formats. This table does not represent the complete list of supported formats; it represents the list of commonly used formats.

See **Object Formats** in the MTP Specification, Revision 1.0 or later for complete list of defined Device Properties.

(2) Support for the following object properties is mandatory for AbstractAudioAlbum objects:

- Genre (0xDC8C)
- Album Artist (0xDC9B)
- WMPMetadataRoundTrip (0x9201)

A device that supports both the AbstractAudioAlbum format and an image format may optionally support album art. Album art is attached to an AbstractAudioAlbum object by adding the art to the album's RepresentativeSampleData property.

If a Portable Device supports album art, then the device must support the following object properties:

- PurchaseFlag (0xd901), an object property in the Windows Media DRM 10 for Portable Devices MTP Extensions
- RepresentativeSampleFormat (0xdc81)
- RepresentativeSampleSize (0xdc82)
- RepresentativeSampleHeight (0xdc83)
- RepresentativeSampleWidth (0xdc84)
- RepresentativeSampleData (0xdc86)
- User Rating (0xdc8a)
- WMPMetadataRoundTrip (0x9201)

Design Notes:

- "Picture Transfer Protocol (PTP) for Digital Still Photography Devices," Version 1.0 of the PIMA15740: 2000 Picture Transfer Protocol specification.
- The Media Transfer Protocol Specification, Revision 1.0 is available at http://go.microsoft.com/fwlink/?LinkId=243145.
- Additional implementation details can be found in the Windows 7 Portable Device Enabling Kit for MTP, which is available at http://go.microsoft.com/fwlink/?LinkId=243146.

Send comments about this topic to Microsoft

Device.Portable.MobilePhone

MobilePhone

In this topic:

Device.Portable.MobilePhone.MTP

Mobile phones must support MTP operations and properties as defined in the Media Transport Protocol revision 1.0 or later, along with specific object formats per device type.

Applies to	Windows 10 x64 Windows 10 x86

Description

Media Transfer Protocol (MTP) is an extension of the Picture Transfer Protocol (ISO 15740). Both specifications include optional commands and operations, but some of these are required for certification. There is a core set of operations and device properties that must be met by each device category in order to be compliant with Windows. Implementation details for MTP are defined in the Media Transfer Protocol Specification, Revision 1.0 or later and will be used as the reference for certification of all portable devices as part of the Windows Hardware Certification Program.

Required Operations

This core set of MTP operations and device properties must be met by each portable device in order to be compliant with Windows. Implementation details for each operation and device property are defined in the Media Transfer Protocol Specification, Revision 1.0 or later.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
GetDeviceInfo	0x1001	R	R	R	R	R
OpenSession	0x1002	R	R	R	R	R
CloseSession	0x1003	R	R	R	R	R
GetStorageIDs	0x1004	R	R	R	R	R

GetStorageInfo	0x1005	R	R	R	R	R
GetNumObjects	0x1006	R	R	R	R	R
GetObjectHandles	0x1007	R	R	R	R	R
GetObjectInfo	0x1008	R	R	R	R	R
GetObject	0x1009	R	R	R	R	R
GetDevicePropDesc	0x1014	R	R	R	R	R
GetDevicePropValue	0x1015	R	R	R	R	R
DeleteObject	0x100B	R	R	I	I	I
SetDevicePropValue	0x100A	R	R	I	I	I
SendObjectInfo	0x100C	R	R	I	I	I
SendObject	0x100D	R	R	I	I	I
GetPartialObject	0x101B	R	R	I	I	I
GetObjectPropsSupported	0x9801	R	R	I	I	I
GetObjectPropDesc	0x9802	R	R	I	I	I
GetObjectPropValue	0x9803	R	R	I	I	I
SetObjectPropValue	0x9804	R	R	I	I	I
GetObjectReferences	0x9810	R	R	I	I	I
SetObjectReferences	0x9811	R	R	I	I	I

Where:

R = Required

I = If Implemented (Optional)

N/A = Not Applicable

See **Operations** in the MTP Specification, Revision 1.0 or later for complete list of defined MTP Operations.

The following operations are highly recommended, though not required:

- FormatStore (0x100F)
- GetObjectPropList (0x9805)
- SetObjectPropList (0x9806)
- GetInterDependentPropDesc (0x9807)
- SendObjectPropList (0x9808) - For this command, your device must also support specification of destination, which allows the MTP initiator to dictate a parent handle for that object.

Required Events

The following events must be supported for all objects:

- ObjectAdded (0x4002)
- ObjectRemoved (0x4003)

If the device supports removable media, it must also support the following events.

- StoreAdded (0x4004)
- StoreRemoved (0x4005)

Operation Responses

An appropriate response must be returned for any and all operations. If an error is encountered, error response codes are also defined and must be provided by the device. For more information, see the "Responses" section of the MTP specification.

Required Device Properties

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Battery Level	0x5001	I	I	I	I	I
Synchronization Partner	0xD401	I	I	I	I	I
Device Friendly Name	0xD402	R	R	I	I	I

Device Icon	0xD405	I	I	I	I	I

See **Device Properties** in the MTP Specification, Revision 1.0 or later for a complete list of defined Device Properties.

Device Icon is not required but is strongly recommended. High-resolution images of the device can be used and exposed in the Windows UI.

Object Formats

The following table represents the list of required object formats for a portable device.

Each object format also has a list of object properties that must be supported per object type.

	Property Code	Mobile Phone	Media Player	Digital Camera	Digital Video Camera	Other (Base)
Undefined	0x3000	I	I	I	I	I
Association	0x3001	R	R	R	R	R
Abstract Audio Album	0xBA03	I(2)	I(2)	N/A	N/A	I(2)
Abstract Audio & Video Playlist	0xBA05	I	I	N/A	N/A	I
WAV	0x3008	R(1)	R(1)	I	I	I
MP3	0x3009	R(1)	R(1)	I	I	I
AVI	0x300A	R(1)	R(1)	I	R(1)	I
MPEG	0x300B	R(1)	R(1)	I	R(1)	I
ASF	0x300C	R(1)	R(1)	I	R(1)	I
EXIF/JPEG	0x3801	I	I	R(1)	I	I
TIFF/EP	0x3802	I	I	R(1)	I	I
BMP	0x3804	I	I	R(1)	I	I

JPEG XR	0xB804	I	I	R(1)	I	I
WMA	0xB901	R(1)	R(1)	I	I	I
AAC	0xB903	R(1)	R(1)	I	I	I
WMV	0xB981	I	I	I	R(1)	I
MP4 Container	0xB982	I	I	I	R(1)	I
3GP Container	0xB984	I	I	I	R(1)	I
3G2	0xB985	I	I	I	R(1)	I
AVCHD	0xB986	I	I	I	R(1)	I

(1) Portable devices that are capable of playing audio and/or video must support at least one of these formats. This table does not represent the complete list of supported formats; it represents the list of commonly used formats.

See **Object Formats** in the MTP Specification, Revision 1.0 or later for complete list of defined Device Properties.

(2) Support for the following object properties is mandatory for AbstractAudioAlbum objects:

- Genre (0xDC8C)
- Album Artist (0xDC9B)
- WMPMetadataRoundTrip (0x9201)

A device that supports both the AbstractAudioAlbum format and an image format may optionally support album art. Album art is attached to an AbstractAudioAlbum object by adding the art to the album's RepresentativeSampleData property.

If a Portable Device supports album art, then the device must support the following object properties:

- PurchaseFlag (0xd901), an object property in the Windows Media DRM 10 for Portable Devices MTP Extensions
- RepresentativeSampleFormat (0xdc81)
- RepresentativeSampleSize (0xdc82)
- RepresentativeSampleHeight (0xdc83)

- RepresentativeSampleWidth (0xdc84)
- RepresentativeSampleData (0xdc86)
- User Rating (0xdc8a)
- WMPMetadataRoundTrip (0x9201)

Design Notes:

- "Picture Transfer Protocol (PTP) for Digital Still Photography Devices," Version 1.0 of the PIMA15740: 2000 Picture Transfer Protocol specification.
- The Media Transfer Protocol Specification, Revision 1.0 is available at http://go.microsoft.com/fwlink/?LinkId=243145.
- Additional implementation details can be found in the Windows 7 Portable Device Enabling Kit for MTP, which is available at http://go.microsoft.com/fwlink/?LinkId=243146.

Send comments about this topic to Microsoft

Device.Storage.Controller

General feature that applies to all storage controllers

In this topic:

Device.Storage.Controller.BasicFunction

Storage controller basic functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Controller Basic Functionality

PCI-based host controllers and adapters must support PCI bus mastering as the default setting, and virtual direct access memory (DMA) services must be supported in the host adapter option ROM.

Devices that cannot perform their function without other registry modifications, other than those performed automatically by the device class co-installer, are not eligible for the certification.

All commands must be passed to the underlying physical device unless the controller is a RAID adapter.

If a device returns less data than requested, it must correctly indicate an underrun condition and adapters must handle this in accordance with the WDK (adjust DataTransferLength).

Non-RAID Controller

Miniport drivers other than RAID drivers may not create pseudo devices to be used as targets for management commands for the adapter when no actual LUNs are present. Instead, a SCSI miniports INF must specify the CreateInitiatorLu parameter under the services Parameters key and set this DWORD value to 1. This may not be done using a coinstaller. Storage miniport drivers do not use this parameter as the adapter may always be used even if no devices are present. Values for storage drivers are documented in the WDK.

The following Storage Controller Driver Logo requirement is for the storage controller driver that does not have a matched submission category in the current Windows Hardware Certification Program. Any storage controller with a matched submission category shall be submitted under its matched category.

- Storage controller driver must be a STORPORT MINIPORT driver.
- Registry Entries for STORPORT Miniport Drivers - STORAGE_BUS_TYPE must set to BusTypeUnknown (0x00).
- For storage controllers, the controller itself must correctly translate the commands and respond in accordance with the applicable SCSI specifications, even if the controller implements other command protocols on a different interface type such as SATA, NVMe, or PQI. Any commands that are not recognized must result in a SCSI check condition with valid sense data.

Device.Storage.Controller.ClassCode

Bus-attached controllers must implement the correct class/subclass code as defined in the PCI Code and ID Assignment Specification (Rev. 1.6).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Controller Class Code

Storage host controllers and adapters must meet the requirements for the device protocol used and any requirements related to the device storage bus.

Bus-attached controllers must implement the correct class/subclass code as specified in PCI 2.3, Appendix D. This applies to all bus types including, but not limited to, IDE, Fibre Channel, SCSI, and SATA-based devices. Any device that implements RAID functionality regardless of whether the RAID implementation is done in hardware, firmware, or in the driver code, must implement the PCI RAID Class Code (01/04) and not use the interconnect class code (for example, a SATA RAID controller must implement the 01/04 class code and not the AHCI class code 01/06/01).

Non-PCI attached storage host controller does not need to report PCI class code. However, it must report the equivalent ACPI Compatibility ID.

Device.Storage.Controller.InfFile

All host adapters must be installed by using Plug and Play mechanisms and require the use of an INF file.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Controller INF File

All host adapters must be installed by using Plug and Play mechanisms and require the use of an INF. Installation programs must use INF files, must pass the most current Chkinf tool, and can explicitly modify the registry values only by using INF file constructs that meet the Windows certification program requirements. Changes can be made only under the following keys:

- TimeoutValues under the class driver services keys (Disk and Tape).
- SpecialTargetList only for SCSIport implementations.
- Device's service key (must be HKLM\System\CurrentControlSet\Services\ and the Parameters\Device subkey under that).
- Signal BusChangeDetected on any link transition or detection of a hot-plug event. A limited settling is allowed before this is signaled, but it must be settable through a registry parameter.
- Implementation of the BusType registry DWORD to correctly set the interface type in accordance with the enumeration in NTDDSTOR.H (see the WDK). This value must be set in the miniport's INF under the service's Parameters key. There is no programmatic way to set it and you may not rely on coinstallers as they do not run under all scenarios.

Device.Storage.Controller.MiniportDriverModel

Storage Miniport Driver Model

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Miniport Driver Model

Drivers used with storage controllers must follow the miniport driver model. Storage Miniports must comply with all miniport interface definitions as defined in the WDK. Fibre Channel, iSCSI, SAS, SAS RAID SATA, SATA RAID, SCSI, and SCSI RAID controller drivers must be STORPORT miniports. Monolithic, full-port drivers or other types of drivers that do not follow the miniport model are not eligible for the certification. All drivers for physical hardware must be implemented to support Plug and Play. Legacy drivers are no longer supported.

Any device that depends on a filter driver for physical disk drive functionality is not eligible for certification. Filter drivers may not be used to bypass any part of the storage stack. For example, a filter driver many not directly access any hardware (such as by using HAL calls) and filter drivers may not be used to link cache manager to the hardware implementation. Filter drivers may not be used to violate any terms of the certification program.

Multipathing drivers may not be tied to specific HBAs except for PCI RAID controllers and must use the MPIO model.

Transient or pseudo-devices may not be exposed to the system. Drivers that specify NODRV may be used to "claim" management devices that report as processor, controller, or MSC device types. Such drivers that do not refer to a service entry are not eligible for the certification, but they can be signed.

Send comments about this topic to Microsoft

Device.Storage.Controller.Ata

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Ata.Interface

PATA Controller Interface

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

PATA Interface

PATA controllers must comply with the ATA/ATAPI-7 specification. Bus-mastering DMA capability is required for all PATA controllers with the exception of compact flash and similar flash-RAM device.

The following requirements are also applied to ATA/ATAPI controllers.

- The PACKET command protocol as defined in ATA/ATAPI-7 Volume 2, Section 11.8, must not be implemented in ATA-only controllers.
- PATA controllers that support the PACKET command protocol must be fully implemented as defined in ATA/ATAPI-7 Volume 2, Section 11.8.
- Identify Device data fields (61:60) and (103:100) must not be used to determine 28-bit or 48-bit LBA addressing support. Instead, bit 10 of word 83 and bit 10 of word 86 must be checked for 48-bit LBA addressing support as defined in ATA/ATAPI-7, Volume 1, Section 4.2.1.

Send comments about this topic to Microsoft

Device.Storage.Controller.Boot

Defines requirements that must be met if the storage controller supports boot

In this topic:

Device.Storage.Controller.Boot.BasicFunction

If the controller implements boot support, it must support Int13h functions.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Controller Boot Support

Option ROMs in host controllers and adapters for any interface type, including RAID controllers, that provide boot support must fully support extended Int13h functions (functions 4xh) as defined in BIOS Enhanced Disk Drive Services - 3 [T13-D1572], Revision 3 or later. Logical block addressing is the only addressing mechanism supported.

It is recommended that controllers also support booting using the Extensible Firmware Interface (EFI) and implement device paths as defined in EDD-3. Starting from Windows 8, it is required for controllers to support booting using the Extensible Firmware Interface (EFI).

SD/eMMC/NAND flash controllers do not have Option ROM, so the first part of this requirement does not apply. EFI support is required.

Device.Storage.Controller.Boot.BitLocker

BitLocker must not cause failure in SAN Boot through storage controllers.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

BitLocker must be properly enabled to protect an operating system in a SAN Boot configuration.

Business Justification

(1) When a server is placed in an environment without adequate physical security, BitLocker protects data on the server against unauthorized access if a server is stolen; (2) When hosting service providers repurpose or decommission storage arrays, BitLocker Disk Encryption prevents data breach.

Send comments about this topic to Microsoft

Device.Storage.Controller.Fc

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Fc.Interface

Fibre Channel HBA interface

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Fibre Channel Interface

Fibre Channel host bus adapter drivers must support the WMI classes and methods required to implement the FC-HBA or SM-API by using the Microsoft HBAAPI.DLL. Vendors may not replace the Microsoft-provided version of the HBAAPI.DLL file. A subset of Hbapiwmi.mof WMI classes and methods are required for Windows compatibility. Other WMI classes are optional and are grouped to form feature sets. If a driver implements any part of an optional feature set, all related classes in that feature set must be supported. In some cases, some features are grouped into subfeatures. If a driver implements such a subfeature, the driver must correctly support that specific subfeature.

Send comments about this topic to Microsoft

Device.Storage.Controller.Fc.NPIV

NPIV is used to deliver Virtual Fibre Channel functionality in Hyper-V.

In this topic:

Device.Storage.Controller.Fc.NPIV.BasicFunction

Hyper-V Virtual Fibre Channel N_Port IO virtualization support

Applies to	Windows Server 2016 Technical Preview x64

Description

N_Port IO Virtualization (NPIV) is the underlying technology used in the Hyper-V feature Virtual Fibre Channel. These tests allow vendors to self-test drivers for compatibility with Virtual Fibre Channel at an API level during the development cycle, and before submitting updated drivers to Microsoft for certification.

The following classes are required: MSFC_VirtualFibrePortAttributes, MSFC_FibrePortNPIVAttributes, MSFC_FibrePortNPIVMethodsEx, MSFC_FibrePortNPIVCapabilities, MSFC_NPIVCapabilities, MSFC_NPIVLUNMappingInformationEx. Optionally, MSFC_DH_Chap_Parameters may be implemented.

These tests cover typical valid and invalid API calls, but do not cover in-depth scenarios covering VM workloads, storage target functionality and data integrity, performance, or other considerations.

Send comments about this topic to Microsoft

Device.Storage.Controller.Fcoe

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Fcoe.Interface

Fibre Channel over Ethernet host bus adapter

Applies to	Windows Server 2016 Technical Preview x64

Description

Fibre Channel over Ethernet host bus adapter

- FCoE adapter must implement FC-BB-5 FC-BB_E specification.
- FCoE adapter miniport must be implemented as a Storport miniport.
- FCoE adapter miniport must define VER_FILEDESCRIPTION_STR and contain the substring "[FCoE]".
- For FCoE adapter miniport INF's [service-install-section]:
  - "DisplayName" entry value is required and must contain the substring "[FCoE]".
  - "Description" entry value is optional, if specified, must contain the substring "[FCoE]".
- For FCoE adapter miniport INF's Models Section [models-section-name] | [models-section-name.TargetOSVersion]:
  - "device-description" entry value must contain the substring "[FCoE]".

- FCoE adapter miniport must declare BusTypeFibre as its STORAGE_BUS_TYPE in the INF file. STORAGE_BUS_TYPE Enumeration
- FCoE adapters that expose PCI storage device(s)/function(s) for FCoE frame processing (either egress or ingress), must report 0x0c0400 (Fibre Channel) as its Class Code (Base Class, Sub-Class and Interface code).

Device.Storage.Controller.Fcoe.Interoperability

Fibre Channel over Ethernet host bus adapter – Interoperability

Applies to	Windows Server 2016 Technical Preview x64

Description

Interoperability

- FCoE adapter must interoperate with FC SAN devices through FCF.
- FCoE adapter must interoperate with native FCoE devices.
- FCoE adapter must be able to transport network (IP) and storage (FCoE) traffic concurrently.
- Disable/removal/loss of FC (storage) functionality must not impact network (Ethernet) connectivity and functionality.
- FCoE adapter must be able to access and address FC and native FCoE devices concurrently (through FCF).

Initiator Coexistence

- FCoE adapter must coexist with FC adapter without interference on the same system.
- FCoE adapter must coexist with other FCoE adapters without interference on the same system.

Send comments about this topic to Microsoft

Device.Storage.Controller.Flush

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Flush.BasicFunction

Flush to connected device

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Industry standard spec requirements:

- For ATA device, FLUSH CACHE (E7h, Non-Data) 28-bit command is optional for ATA devices and ATAPI devices. FLUSH CACHE EXT (EAh, Non-Data) 48-bit command is mandatory for devices implementing the 48-bit Address feature set.
- For SCSI Devices, SYNCHRONIZE CACHE (10) command and /or SYNCHRONIZE CACHE (16) command shall be implemented.

Windows design spec requirements - controller:

- For controllers and device drivers of all bus types (ATA, SCSI and USB), flush cache command shall be sent to connected device without any omission.

Send comments about this topic to Microsoft

Device.Storage.Controller.Iscsi

Defines the industry and Microsoft standards that must be met

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) Windows Server 2016 Technical Preview x64

In this topic:

Device.Storage.Controller.Iscsi.Interface

iSCSI interface

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

iSCSI Interface

iSCSI host bus adapters must be compatible with iSCSI RFC3720 and must implement all mandatory requirements. The only exception to this is that IPsec is not mandatory but, if implemented, will be tested. All optional components, if implemented, must comply with this specification.

- Optional behavior must not undermine compliance with the iSCSI specification or the Windows certification program requirements for iSCSI.
- The device and drivers must also meet applicable requirements for SCSI controllers and devices.
- An iSNS client must be implemented and comply with RFC3723.
- An adapter must be able to receive ping (ICMP) and send ping (ICMP).
- Device logons must be consistent with the Microsoft iSCSI Discovery Service.
- Any boot device configured by other means must be reported to the service after boot.
- Any other persistent target assignments and sessions under control of the HBA must be reported by using WMI to the iSCSI Initiator Service when it is available.
- The following logon authentication implementations are both required:
  - "CHAP-target authenticates initiator"
  - None
- Mutual CHAP, if implemented, must adhere to the specification and will be tested.
- IPSec support must adhere to all applicable IPSec requirements in this document.

- The HBA driver must implement all required WMI interfaces documented in the WDK.
- The initiator must perform an automatic logon to targets assigned to the computer as persistent targets. The initiator will connect to all persistent targets before the targets are enumerated by Windows, which starts with an Inquiry to LUN 0. If a connection drops, it will continue to try to reconnect. Devices cannot depend on the discovery service for this information.
- Initiators must:
  - Maintain the persistent logon information in the registry or in NVRAM.
  - Support the new WMI class for defining/managing persistent logons.
  - Persist IP network adapter and discovery configurations (IP configuration information, such as static IP address, static default gateway IP address, static subnet mask, and DNS server) or use DHCP to obtain this information.
  - For discovery configuration, remember which discovery methods are used and, for iSNS, maintain the address of the iSNS server.
- An iSCSI HBA must support changers, disk, tape, and external RAID devices.

Send comments about this topic to Microsoft

Device.Storage.Controller.Iscsi.iSCSIBootComponent

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Iscsi.iSCSIBootComponent.FwTable

iSCSI Boot functionality

Applies to	Windows Server 2016 Technical Preview x64

Description

iSCSI Boot functionality

The iSCSI boot component must:

- Be compatible with iSCSI targets complying to iSCSI RFC3720.
- Defer all iSCSI functionality to the Microsoft iSCSI software initiator after the Windows boot sequence begins (once ntoskrnl.exe loads).
- Support Login Redirection.
- Support one-way CHAP and must maintain CHAP secret in non-volatile memory.
- Implement iSCSI Boot Firmware Table in firmware or Option ROM per the ACPI SIIG optional table.
- Support crashdump.
- Support the use of IPV6 addressing.
- Support iSNS (Internet Server Name Service).
- Support RELOGIN as minimum error handling in the case of an error during logon initialization sequence with the iSCSI target.
- Support the following DHCP option numbers: 1, 3, 17, 43, 51, 54, 57, 60, 61, 255, 201, 202, 203, 204, 205, and 206.
- Support the following DHCP parameters:
- DHCPDISCOVER to get an IP address assigned.
- DHCPREQUEST to obtain iSCSI protocol parameters.
- DHCPINFORM to inquire updates in information from the DHCP server.

In addition:

- All requests from the iSCSI preboot component DHCP clients to DHCP servers must use option 60 to signal the appropriate vendor scope.
- All requests from the iSCSI preboot component DHCP clients to DHCP servers must use option 61 to signal the identity of this given client. If the client Alt ID is not defined, then the type field should be set to 0x01 and the EN MAC address must be used to define client identity. If the client Alt Id is defined, then the type field should be set to 0x00 and the CAID field must be used.

- All requests from the iSCSI preboot component DHCP clients to DHCP servers must use the CHADDR field containing the EN MAC address of the DHCP client.
- The use of the CIADDR in iSCSI preboot component must conform to the DHCP usage of CIADDR.
- The use of the YIADDR iSCSI preboot component must conform to the DHCP usage of YIADDR; namely, the iSCSI preboot component DHCP client must accept the YIADDR provided by the DHCP server during the DHCPREQUESTÃ³DHCPACK or DHCPINFORMÃ³DHCPACK transaction sequence.
- The use of SIADDR in the iSCSI preboot component must conform to the DHCP usage of SIADDR; namely, the iSCSI preboot component DHCP client must use this address to access the DHCP server during DHCPREQUESTÃ³DHCPACK or DHCPINFORMÃ³DHCPACK transactions.
- To support DHCP option 1, the subnet mask provided in the DHCPOFFER response from the iSCSI pre-boot component must provide the subnet mask.
- All transactions between the iSCSI preboot component DHCP clients and DHCP servers must be a single-frame transaction.
- To avoid conflict with other services, the iSCSI preboot component must not use DHCP option 52.
- Implementation of multiple option responses in the iSCSI preboot component must comply with RFC 3396.

Send comments about this topic to Microsoft

Device.Storage.Controller.Optical

General feature that applies to all storage controllers to ensure optical burning requirements are met.

In this topic:

Device.Storage.Controller.Optical.BasicFunction

Storage HBA drivers must support optical drives.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Controller optical support

The storage HBA drivers must support the optical device. The CDBs sent to the optical device and the response from the optical device must be handled properly.

Send comments about this topic to Microsoft

Device.Storage.Controller.PassThroughSupport

Pass-through storage controller basic functionality

In this topic:

Device.Storage.Controller.PassThroughSupport.BasicFunction

Pass-through storage controller basic functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The bus adapter must report the actual bus type used to connected drives (e.g. drives are connected via the SAS bus; reporting drives as connected via the "RAID" bus is invalid). All commands must be passed directly through to the underlying physical devices. The physical devices must not be abstracted (e.g. formed into a logical RAID disk) and the bus adapter must not respond to commands on behalf of the physical devices.

NOTE: This applies only to SAS and SATA controllers.

Send comments about this topic to Microsoft

Device.Storage.Controller.Raid

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Raid.BasicFunction

RAID controller

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

RAID Controller

- Miniport drivers for PCI RAID adapters may create a management LU if this device is always available for management commands and has a device type other than disk. Since this device will appear in device manager, a NODRV INF may be submitted to claim this device and prevent user popups (this INF may be signed).
- For RAID controllers, the controller itself must correctly interpret the commands and respond in accordance with the applicable SCSI specifications, even if the controller implements RAID on a different interface type such as SATA. Any commands that are not recognized must result in a SCSI check condition with valid sense data.

SCSI Requirements can be found in the Device.Storage.SCSI section.

Send comments about this topic to Microsoft

Device.Storage.Controller.Raid.ContinuousAvailability

Validates that Continuous Availability (CA) storage controller and drivers meet all applicable requirements

In this topic:

- Device.Storage.Controller.Raid.ContinuousAvailability.LunAccess
- Device.Storage.Controller.Raid.ContinuousAvailability.RAID
- Device.Storage.Controller.Raid.ContinuousAvailability.RecoveryProcessing

Device.Storage.Controller.Raid.ContinuousAvailability.ActiveMode

A device and its driver must support active LUN access on each node in a Windows Failover Cluster system configuration.

Applies to	Windows Server 2016 Technical Preview x64

Description

When the device and driver are configured in multiple nodes in a Windows failover cluster, each node must support as a minimum “dual-active” access, defined as full read/write functionality for at least one LUN that has the capability to be failed over to another node in the cluster. Specifically, a device is not allowed to only support passive operation where the device and driver operate only in the standby mode until the cluster fails over a LUN from another node.

Design Notes:

- For this “dual-active” access, LUNs do not need to provide full, performant read/write functionality to all nodes in the cluster. Access from other nodes only needs to be supported as required to support a valid Windows failover cluster.
- Specifically, it is desirable but not required to support simultaneous, shared access to a LUN from multiple nodes in the cluster.

Device.Storage.Controller.Raid.ContinuousAvailability.FailoverClustering

A device and its driver must meet a minimum set of requirements to operate in a Windows failover cluster.

Applies to	Windows Server 2016 Technical Preview x64

Description

A device must comply with SPC-3 section 6.11 PERSISTENT RESERVE IN command

- A device must comply with SPC-3 section 6.12 PERSISTENT RESERVE OUT command
    - Device must support Persistent reservations type as defined in section 6.11.3.4 Table 107:
        - 5h Write Exclusive – Registrants only
- A device must comply with Reservation behavior as defined in SPC-3 section 5.6

- Specifically, SCSI compliance testing should take special care to verify the following:
  - Verify Reserving – once reserved, the scope & type of an existing persistent reservation cannot be changed
  - Comply with section 5.6.1 – table 32
  - Comply with section 5.6.6 – table 33, table 34, table 36 – verify Good Status conditions
  - Verify, re-register an I_T nexus that has already been registered (5.6.6) – should return good status
  - Verify, re-reserve an I_T nexus that holds the persistent reservation (5.6.9) – should return good status
  - Return Reservation conflict – if PRO command with service action PREEMPT or PREEMPT & ABORT is sent with invalid Service Action Reservation Key (5.6.10.4.4)
  - Verify Additional Length Field per 6.11.3.2 - should return zero when no persistent reservation is held

Design Notes:

- It is recommended that in addition to the standard HCT qualification, all solutions are also validated with the "Microsoft Cluster Configuration Validation Wizard" (ClusPrep) tool.
- Only SAS devices using the Serial SCSI Protocol (SSP) transport will be supported on failover clusters (including SAS JBOD or any SAS SSP RAID systems).

If the system disks are attached to a bus type that is not a valid type for shared storage (something other than FC, iSCSI, or SAS), then the system disks and shared.

Device.Storage.Controller.Raid.ContinuousAvailability.LunAccess

A device and its driver must meet requirements for accessing LUNs in a Windows failover clustering configuration.

Applies to	Windows Server 2016 Technical Preview x64

Description

When a node in a Windows Failover Cluster fails over its resources, either planned or unplanned (e.g., resulting from a node crash or node power failure), the device and driver must meet the following requirements:

- All data from successfully acknowledged writes prior to failover must be preserved and accessible from surviving nodes. Specifically, data corresponding to write operations sent to the device and driver on the failing node prior to the start of failover of its LUNs (i.e., after the corresponding RAID controller on a surviving node receives a persistent reservation request corresponding to the LUNs from the failed node) that have been acknowledged by the driver as completed must be accessible after failover completes from a surviving node. In addition, any data that may be pending from unacknowledged writes at the start of failover must not change data subsequently written to the device from a surviving node.
- When the LUNs supported by the RAID controller on the failing node stop being accessible (i.e., stop acknowledging and processing I/O requests), the LUN must be made accessible (i.e., acknowledge and process I/O requests) by at least one other node in the cluster with a maximum delay of 5 seconds. This time limit must be supported for up to 100 LUNs or the maximum number of LUNs supported by the controller, whichever is smaller.

During normal operation (i.e., not during failover) in a Windows failover cluster, the device and driver must meet the following requirement:

- All I/O requests to any LUN supported by the RAID controller must complete within 25 seconds. This time limit must be supported for up to 100 LUNs or the maximum number of LUNs supported by the controller, whichever is smaller.

Device.Storage.Controller.Raid.ContinuousAvailability.RAID

A device and its driver must meet a minimum set of requirements for RAID functionality.

Applies to	Windows Server 2016 Technical Preview x64

Description

- A device and its driver must support commands from SBC-2 (or later) regardless of the drive interface implemented.
- A device and its driver must support, at a minimum, one of: RAID1, RAID 5, RAID6 or RAID 1/0, or an equivalent feature that supports full LUN functionality in the event of failure of a single data drive.
- A device and its driver RAID implementation must provide a solution to prevent data loss due to the RAID “write hole” failure mechanism. (See Design Notes for definition of the RAID “write hole” failure mechanism.)

Design Notes:

If there is a system or controller failure during active writes, the erasure code information of a stripe (e.g., parity) may become inconsistent with the data. Data loss may result if this incorrect erasure code information is subsequently used to reconstruct the missing block in that stripe. This problem is known as the RAID “write hole”.

- Examples of typical solutions for the RAID write hole problem are to provide mechanisms to detect and recover from an interrupted update-in-place operation or to avoid update-in-place semantics.

Device.Storage.Controller.Raid.ContinuousAvailability.RecoveryProcessing

A device and its driver must automatically complete all recovery processing resulting from failing over a LUN from a node in a Windows failover cluster.

Applies to	Windows Server 2016 Technical Preview x64

Description

When the device and driver are configured in a Windows failover cluster and LUN access is restored after a failover (see requirement CAHWStorage-0004), all processing required by the device and driver to recover from the failover operation and restore normal operation must complete automatically, i.e., without any manual intervention.

Send comments about this topic to Microsoft

Device.Storage.Controller.Sas

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Sas.Interface

SAS controller interface

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

SAS Interface

SAS host bus adapter miniport drivers must use the Microsoft hbaapi DLL to support the Windows Management Instrumentation (WMI) methods. The specific required WMI classes and methods are grouped and designated as mandatory or optional. All mandatory classes and methods must be supported. If a group is identified as optional and a miniport driver supports that group, individual methods and classes within that group are also classified as mandatory if the group is implemented, or optional, if the group is not implemented.

Note: The SAS HBA API is currently in the draft stage at the T11.5 working group. This support will not be a requirement until the draft document is complete. WHQL will issue an announcement when this support becomes a requirement.

Send comments about this topic to Microsoft

Device.Storage.Controller.Sata

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.Sata.Interface

SATA controller interface

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

SATA Interface

- SATA controllers must comply with the ATA/ATAPI-7 specification.
- SATA controllers must support hot plug.
- SATA controllers shall support Asynchronous Notification as defined in Serial ATA: High Speed Serialized AT Attachment, Version 2.6 or later and AHCI 1.3 or later.
- If implemented, NCQ must be supported properly.
- If implemented, larger sectors other than 512 bytes must be supported properly.

- AHCI SATA controllers must comply with the AHCI 1.0 specification or later.
- SATA controllers shall not emulate PATA.
- Interfaces for non-AHCI SATA controllers, if implementing an interface other than AHCI, must be supported by a Windows inbox driver or must certify with a supplied driver set. The supplied drivers must meet the driver certification requirements in this document.
- Recommendation: SATA controllers should implement interface power management.
- Recommendation: SATA controllers should implement Native Command Queuing (NCQ) support.

Send comments about this topic to Microsoft

Device.Storage.Controller.SD

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Controller.SD.BasicFunction

SD controller basic functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- Supports the SD Standard Host Interface (as per the SD 3.0 Standard).
- Supports CMD21 tuning for eMMC HS200.
- All interrupts shall be disabled except for buffer read ready when executing the tuning procedure for either SD 3.0 devices (CMD19) or eMMC 4.5 devices (CMD21).
- Tuning procedure must use the standard tuning blocks defined in the SD and eMMC standard specifications, respectively.

- Supports 1.8V and 3.3V signaling voltages.
- Supports ADMA2 (no system DMA).
- Properly express all capabilities supported by the host controller in the standard capabilities register.
- Supports Byte (8-bit), Word (16-bit), and Double Word (32-bit) register accesses based on the register size given in the standard host controller specification.
- Supports write protect where write protect is indicated by the value 0.
- Supports 1-bit and 4-bit bus widths. 8-bit bus width is also required for eMMC controller.
- Supports all UHS-I modes (SDR50, DDR50, SDR104). HS200 is also required for eMMC controller.
- No retuning shall be necessary for SDR50 mode.
- Retuning shall not require a software timer.
- Supports Auto CMD23 and Auto CMD12.
- No toggling of any proprietary register bits shall be necessary to enable any functionality.
- Clock frequency shall be calculated according to standard specification.
- Support standard error recovery procedure.

Send comments about this topic to Microsoft

Device.Storage.ControllerDrive.NVMe

Storage NVMe feature

In this topic:

Device.Storage.ControllerDrive.NVMe.BasicFunction

NVMe device requirements

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

Description

The following requirements must be fulfilled by the NVMe device in general:

- The device must support PCIe Gen2 or higher.

The device must support the following interrupt modes:

- Running with the requested number of interrupts (i.e. > 1 MSI or MSI-X based interrupt)
- Running with only 1 MSI or MSI-X interrupt
- Running with only 1 line-based interrupt
    - [Server] MSI-X is required
    - [Client] MSI or MSI-X is required

The device must use the following class and sub-class codes for identification, for further driver matching the vendor and product IDs should be utilized:

- Base Class:01h
- Sub-Class:08h
- Programming Interface:02h

The device must have at least one upgradeable firmware slot.

[Client] [If Implemented] The device shall support either:

- A non-operational NVMe power state with a maximum listed power consumption of no more than 5mW and an exit latency to an operational state of less than 100ms (implies L1.2), or
- The ability to resume from D3cold (power-off) within 500ms, while not exceeding an entry latency of 1,000ms.

Note: If such a power state or capability is not provided by the device, experience on Connected Standby systems will be affected negatively as battery drain increases significantly.

It is highly recommended that NVMe devices that are to be used in client systems with limited cooling capability support at least the following in addition to 100% on and off states:

- 1 operational power state (reduced performance) – such that significant thermal throttling of the device is possible
- 1 non-operational power state with a resume latency of at most 50ms.

The following requirements that the device must fulfill are specific to revision 1.0 of the official NVMe spec:

- 3.1.1 Controller Capabilities
  - MPSMIN (Memory Page Size Minimum) must be set to 0. Bit: 51:48
- 5.2 Asynchronous Event Request
  - Error events, as well as SMART / Health status events must be supported, see section 5.12 below.
- 5.3 Create I/O Completion Queue
- 5.4 Create I/O Submission Queue
- 5.5 Delete I/O Completion Queue
- 5.6 Delete I/O Submission Queue
- At least 2 queues must be supported for 5.3 through 5.6
  - 1 set of I/O submission and completion queues and 1 set of admin submission and completion queues are acceptable.
  - Note: It is recommended to provide at least 4 I/O queues on client devices and 16 or more on server devices to provide the ability to bind one set of queues per processor on the system.
- 5.7 Firmware Activate
  - Commands arriving during the download or activation period must not be failed, but instead queued for later execution.
  - Activation of a firmware image should be done without requiring a power cycle if possible. A device internal reset is acceptable and expected.
- 5.8 Firmware Image Download
  - The device must not fail I/O during the download phase and shall continue serving I/O.
  - If the device is dual-ported the download operation shall be possible utilizing both ports, i.e. part of the image can be downloaded through port 1, while another part arrives via port 2.
- 5.9 Get Features – at least the following must be reported accurately:

- 5.12.1.5 Error Recovery
- 5.12.1.6 Volatile Write Cache
  - A volatile write cache is not required on the device to adhere to these requirements. This field has to be accurately reported.
- 5.12.1.8 Interrupt Coalescing
  - [Server] required
  - [Client] not required
- 5.12.1.11 Asynchronous Event Configuration
- 5.10 Get Log Page
  - The device must implement and populate at least the log pages for Error Information (01h), SMART / Health Information (02h), and Firmware Slot Information (03h)
- 5.11 Identify
  - MDTS (Maximum Data Transfer Size) must be either 0 (no limitation) or at least 5 (128KB). Byte: 77
    - Note: This value may increase in future revisions and next generation devices should be designed with this in mind.
  - NN (Number of Namespaces) must be at least 1. Bytes: 519:516
  - FLBAS (Formatted LBA Size) must have bit 4 set to 0. Byte: 26
    - If the Metadata Capabilities feature is supported, this requirement holds, otherwise it can be ignored; i.e. iff MC bit 1 is set to 1, the above FLBAS requirement is binding. Byte: 27
  - LBADS (LBA Data Size) must be set to 9 or 12, i.e. 512B or 4KB. Bits: 23:16
    - Other LBA data sizes are acceptable, as long as 512B or 4KB is supported.
- 5.12 Set Features – at least the following must be implemented:
  - Error Recovery (05h)
  - Volatile Write Cache (06h)
    - If a volatile write cache exists on the device

- Number of Queues (07h)
- Interrupt Coalescing (08h)
- Asynchronous Event Configuration (0Bh)

- 5.13 [If Implemented] Format NVM
  - The device should be capable to perform a Cryptographic Erase (SES Value 010b)
    - If cryptographic erase is not supported, Windows will not utilize the FormatNVM command.
  - Format NVM must be targetable at individual namespaces, i.e. Bit 1 and Bit 0 in the Identify Controller data structure, of Byte 524 (FNA), must be set to 0.
- 6.6 Dataset Management – Deallocate
  - All I/O and Deallocate commands shall be completed in less than 500 ms.
  - 98.5% of I/O commands shall be completed in less than 100 ms.
- 6.7 Flush
  - If a volatile cache exists on the device
- 6.8 Read
- 6.9 Write

Send comments about this topic to Microsoft

Device.Storage.Enclosure

Drive enclosures must meet these requirements.

In this topic:

Device.Storage.Enclosure.DirectAccess

Drive enclosures must provide direct access to the drives they house.

Applies to	Windows Server 2016 Technical Preview x64

Description

Enclosures must not abstract the drives they house (e.g. formed into a logical RAID disk). Integrated switches, if present, must provide discovery of and access to all the drives in the enclosure without requiring additional physical host connections.

Device.Storage.Enclosure.DriveIdentification

Drive enclosures must provide a drive identification service.

Applies to	Windows Server 2016 Technical Preview x64

Description

Storage enclosure must meet the following requirements to support storage space configuration.

- Must support the following commands:
  - INQUIRY
  - SEND DIAGNOSTIC
  - RECEIVE DIAGNOSTIC RESULTS
  - REQUEST SENSE
  - TEST UNIT READY
- Must set ENCSERV bit to one in the Standard Inquiry Data (SPC4) to indicate that storage target device contains an embedded enclosure services component that is addressable through this logical unit.
- Must support the following diagnostic pages for enclosure service devices:
  - Supported Diagnostic Pages diagnostic page (00h)
  - Configuration diagnostic page (01h)
  - Enclosure Control diagnostic page (02h)
  - Enclosure Status diagnostic page (02h)
  - Additional Element Status diagnostic page (0Ah)
- The following Type Control and Status Elements are used in Windows Storage Space feature;

Element Type Code	Element Type Name
01h	Device Slot
02h	Power Supply
03h	Cooling
04h	Temperature Sensor
07h	Enclosure Service Controller Electronics
12h	Voltage Sensor
13h	Current Sensor
17h	Array Device Slot

- Storage enclosure must support either Device Slot (01h) or Array Device Slot (17h) Element Type. All other element types (Power Supply, Cooling, Temperature Sensor, Enclosure Service Controller Electronics, Voltage Sensor, and Current Sensor) are optional.
- For Additional Element Status diagnostic page (0Ah), enclosures shall provide additional element status descriptor with the EIP (element index present) bit set to one:
  - The Element Index field that is reported by drive bays are in an ascending order.
  - The protocol identifier must set to 6h (SAS).
- The SES device must report the same address the drive reports for Device Identification VPD page (83h) should include one designation descriptor in which the target port name or identifier (see SAM-5) is indicated. The designation descriptor, if any, shall have the ASSOCIATION field set to 01b (i.e., target port) and the DESIGNATOR TYPE field set to:

  a) 2h (i.e., EUI-64-based);

  b) 3h (i.e., NAA); or

  c) 8h (i.e., SCSI name string).

- For Enclosure Control diagnostic page (02h), control descriptors are enumerated by disk bay number in an ascending order.
- Must comply with T10 SES3 spec to implement both Enclosure Control diagnostic page and Enclosure Status diagnostic page with INFO, 1NON-CRIT, CRIT and UNRECOV bits.
- ELEMENT STATUS CODE field in all Status element formats must adopt the following codes:

Code	Name	Condition
0h	Unsupported	Status detection is not implemented for this element.
1h	OK	Element is installed and no error conditions are known.
2h	Critical	Critical condition is detected.
3h	Noncritical	Noncritical condition is detected
4h	Unrecoverable	Unrecoverable condition is detected.
5h	Not Installed	Element is not installed in enclosure.
6h	Unknown	Sensor has failed or element status is not available.
7h	Not Available	Element is installed, no known errors, but the element has not been turned on or set into operation.
8h	No Access Allowed	The initiator port from which the RECEIVE DIAGNOSTIC RESULT command was not received does not have access to this element.
9h to Fh	Reserved	

Notes: Windows correlates enclosure services to drives via the protocol-specific information and the drives' Device Identification VPD page (83h) with ASSOCIATION field set to 1.

Send comments about this topic to Microsoft

Device.Storage.Hd

In this topic:

Device.Storage.Hd.BasicFunction

HD basic functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The device must be able to perform the following scenarios:

- Sequential read
- Sequential write
- Sequential verify (write followed by read and comparison)
- Random read
- Random write
- Random verify

Device.Storage.Hd.PhysicalSectorSizeReportsAccurately

The reported physical sector size must be the unit of an atomic write.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Following applies to ATA based Hard Disk Drives:

If implemented, support for storage devices with logical sector sizes larger than 512-bytes need to be implemented as described in the ATA-8 specifications. Please refer to the INCITS T13 specification repository for access to the specification.

Following applies to SCSI based Hard Disk Drives:

If implemented, support for storage devices with logical sector sizes larger than 512-bytes need to be implemented as described in the SBC-3, SPC-4, and SAT-3 specifications. Please refer to the INCITS T10 specification repository for access to the relevant specifications.

Business Justification

Some hard disk drives report the physical sector size of the disk incorrectly. For example, the drive is released a "4K" drive without reporting that it is indeed a 4K drive. Applications use the reported physical sector size as a notion of atomicity and perform I/O based on this. The most basic example is a database-style application will only store one commit record within the unit of atomic write for fear of loss if power is lost or if a physical sector becomes physically bad. When the reported physical sector size is not the unit of atomicity, serious reliability concerns can arise in scenarios where power is lost such as: Applications can fail to recover, and users will need to restore from backup. Applications can fail to recover, but the application will need to perform a lengthy consistency check. Corruption of metadata, log file data, user data, or even data from other applications.

Device.Storage.Hd.RotationalRate

Hd rotational rate logo requirements

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Solid State Device (SSD) shall set Nominal Media Rotation Rate to a non-rotating medium value. A rotational or a mixed rotational and SSD device shall report the rotational rate of the rotational medium.

Following applies to ATA based Hard Disk Drives:

ATA hard disk devices must report nominal media rotation rate as described in the ATA-8 specifications. The Word 217 of Identify Device command return shall not be 0000h.

Nominal Media Rotation Rate Value Description

Value	**Description**
0000h	Rate not reported
0001h	Non-rotating media (e.g., solid state device)

0002h-0400h	Reserved
0401h-FFFEh	Nominal media rotation rate in rotations per minute (rpm) (e.g., 7 200 rpm = 1C20h)
FFFFh	Reserved

Following applies to SCSI based Hard Disk Drives:

SCSI hard disk device must report nominal media rotation rate as described in the T10 SBC3 specifications. The Inquiry command return for Block Device Characteristics VPD pageshall not set media rotation rate = 0000h.

MEDIUM ROTATION RATE field

Code	**Description**
0000h	Medium rotation rate is not reported
0001h	Non-rotating medium (e.g., solid state)
0002h-0400h	Reserved
0401h-FFFEh	Nominal medium rotation rate in rpm (e.g., 7 200 rpm = 1C20h,10 000 rpm = 2710h, and 15 000 rpm = 3A98h)
FFFFh	Reserved

Send comments about this topic to Microsoft

Device.Storage.Hd.1394

In this topic:

Device.Storage.Hd.1394.Compliance

IEEE 1394 hard disk drive specification compliance

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

1394 Compliance

IEEE-1394 (Firewire) devices must comply with Serial Bus Protocol-2 (SBP-2) and SCSI Primary Commands-2 (SPC-2), and disk devices must comply with SCSI Reduced Block Commands (RBC).

The reference for specification compliance...:

SBP-2, SPC-2, Min:RBC

Send comments about this topic to Microsoft

Device.Storage.Hd.Alua

In this topic:

Device.Storage.Hd.Alua.Compliance

Asymmetric Logical Unit Acces (ALUA)

Applies to	Windows Server 2016 Technical Preview x64

Description

If a device supports asymmetric logical unit access (ALUA), the device must comply with the requirement of implementing target port group support (TPGS) in standard inquiry data as SPC3-r23 section 6.4.2.

The Report Target Port Group command must be supported, if logical units report in the standard Inquiry Data that they support ALUA. The storage device must comply with SPC3 -r23 section 6.25 Report Target Port Group command according to its TPGS field code in the standard Inquiry data.

Send comments about this topic to Microsoft

Device.Storage.Hd.Ata

In this topic:

Device.Storage.Hd.Ata.BasicFunction

ATA/ATAPI interface

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

PATA Devices (legacy)

(PATA Devices will no longer be accepted for WHQL submission after June 2013.)

Microsoft recommends the use of SATA for new devices. However, in a spirit of compatibility with existing device base, the following requirements are provided for PATA devices.

Shared bus capabilities are required for PATA devices; devices shall be configurable as device 0 or device 1.

Device.Storage.Hd.Ata.Dma

ATA/ATAPI DMA mode.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

ATA/ATAPI DMA Mode

All PATA controllers and PATA peripherals shall support Ultra-DMA as defined in ATA/ATAPI-7.

Justification:

In addition to improved transfer rates, Ultra-DMA also provides error checking for improved robustness over previous PATA implementations.

Send comments about this topic to Microsoft

Device.Storage.Hd.AtaProtocol

In this topic:

Device.Storage.Hd.AtaProtocol.Performance

ATA device performance

Applies to	Windows 10 x64 Windows 10 x86

Description

ATA device performance

The Windows 7 Windows System Assessment Tool (WinSAT) disk formal test for the block storage device must pass the following performance requirements for any visible storage space utilization up to 95% (% of utilization as % of "used space" seen through the Windows file system).

- Disk Sequential 64 K Byte Read >25 MB/s
- Disk Random 16 K Byte Read >0.5 MB/s
- Disk Sequential 64 K Byte Write >20 MB/s
- Average Read Time with Sequential Writes <25 ms
- Latency: 95th Percentile <120 ms
- Latency: Maximum <700 ms
- Average Read Time with Random Writes <40 ms

Device.Storage.Hd.AtaProtocol.Protocol

ATA/ATAPI protocol

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

ATA/ATAPI Protocol

- ATA/ATAPI controllers and devices shall comply with the following standard(s)
  - INCITS 397-2005 (1532D): AT Attachment with Packet Interface - 7 or later, also referred to in this document as ATA/ATAPI-7. AT Attachment with Packet Interface - 8 or later will be required when this revision 8 is final and published.
- ATA/ATAPI controllers shall support Windows operating system boot
- ATA/ATAPI devices shall not rely on Identify Device data fields (61:60) and (103:100) to determine 28 bit or 48 bit LBA addressing support. ATA/ATAPI shall rely instead on bit 10 of word 83 and bit 10 of word 86 to identify 48 bit LBA addressing support (as per ATA/ATAPI-7).

Design notes

Recommended:

Reporting Nominal Media Rotation Rate

If a device requires Windows defragmentation to be turned off by default, the device should report its Nominal Media Rotation Rate as 0001h Non-rotating media (e.g. solid state device) as per the ATA8-ACS1 specification, section 7.16.7.77.

Justification:

When the Nominal Media Rotation Rate reported by the device is anything but 0001h Non-rotating media, Windows will by default perform defragmentation of the block storage device.

Send comments about this topic to Microsoft

Device.Storage.Hd.DataVerification

Disk Verification Tests to ensure there is no data loss or corruption

In this topic:

Device.Storage.Hd.DataVerification.BasicFunction

All storage devices must work correctly on Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Storage devices must reliably read and write data without data loss or data corruption.

Send comments about this topic to Microsoft

Device.Storage.Hd.Ehdd

In this topic:

Device.Storage.Hd.Ehdd.Compliance

Encrypted hard drive complies with Microsoft and industry specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Encrypted hard drive

eDrives must be compliant with these industry specifications

- IEEE 1667 version IEEE 1667-2009
  - Support Probe silo
  - Support TCG Storage silo
- TCG
  - TCG core specification version 2.0
  - OPAL SSC 2.0
    - Programmatic TPer Reset Rev

        - Modifiable CommonName Column
        - SID Authority Disable
    - OPAL SSC Feature Sets
        - Set Additional Data Store Rev 1.05 or later
        - Single User Mode Rev 1.02 or later
- SCSI
    - SPC4
    - SAT2
- ATA
    - ACS2

eDrives must comply with these Windows Design Spec requirements:

- Support at least AES 128
- Support one or more of the following cipher modes
    - CBC
    - XTS
- Support at least 8 bands
- Support additional data store tables
- Support range crossing
- Support authenticate method
- Support secret protect info
- Support modifiable common name
- Support TCG stack reset
- Support programmatic TPer reset
- Support single user mode
- If SCSI devices(SPC4):-

- The 1667 Version Descriptor, 0xFFC2 (IEEE 1667-2009) should be reported in the INQUIRY data. The 'Additional Length' field of the INQUIRY data must be greater than or equal to 0x38.
- Security Protocol IN output must report 00, 01, 02, EE in the Supported Security Protocol List payload

- If ATA devices (ACS2):-
  - The TrustedComputer.FeatureSupported (word 48 - bit 0 must be set to '1') must be reported in the IDENTIFY data
  - The AdditionalSupported.IEEE1667IDENTIFY (word 69 - bit 7 should be set to '1') must be reported in the IDENTIFY data
  - Trusted Receive output should report 00,01, 02, EE in the Supported Security Protocol List payload
- If SATA-USB:-
  - Support SAT2
- Command Performance:- The drive must complete the following operations within the specified duration

Operations	Max completion time
Discovery/Enumeration	24 sec (8 bands)
Activate	45 sec
Revert	8 sec
Create Band	1.5 sec
Delete Band	2 sec
Erase Band	2 sec
Set Metadata	20 sec
Get Metadata	14 sec

Lock Band	1.5 sec
Unlock Band	1.5 sec

Send comments about this topic to Microsoft

Device.Storage.Hd.EMMC

Defines the industry and Microsoft standards that must be met

In this topic:

Device.Storage.Hd.EMMC.BasicFunction

Emmc basic functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Must support industry standards.

eMMC 4.5.1 Requirements

- Support discard/sanitize on both sector size and erase block boundaries.
- Support HPI/BKOPS.
- Support Packed commands.
- Support HS200 signaling.
- Support DDR50 signaling.
- Support a RPMB of at least 128 KB in size and creation of GPPs.
- Support sleep (CMD5).
- Support crypto offload operations (CMD53/54).

- Support OS initiated cache flushing if device supports volatile cache.

Send comments about this topic to Microsoft

Device.Storage.Hd.EnhancedStorage

In this topic:

Device.Storage.Hd.EnhancedStorage.1667Compliance

Enhanced Storage devices must comply with the IEEE 1667 defined standards.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

IEEE1667-Class (Enhanced Storage) enabled storage devices must meet industry standards.

Enhanced Storage devices must comply with the IEEE 1667 defined standards.

- Enhanced Storage device must:
  - Support authenticating the host
  - Implement support for IEEE 1667 (version 1.1 or later) defined Probe Silo on the device.
  - Implement at least one Certificate or Password Silo on the device.
- Enhanced Storage device that implements Certificate Silo must:
  - Load native windows certificate silo driver.
  - Respond to all commands of the IEEE 1667 version 1.1 specification.
  - Verify certificate-based authentication is used to allow and block access to volume.
- Enhanced Storage device that implements Password Silo must:
  - Load native password silo driver.
  - Respond to all commands in the IEEE 1667 Password Silo specification.

- Verify password-based authentication is used to allow and block access to the volume.

Design Notes:

Obtain IEEE 1667 specification from IEEE at the following location:http://go.microsoft.com/fwlink/p/?linkid=110100

Send comments about this topic to Microsoft

Device.Storage.Hd.FibreChannel

In this topic:

Device.Storage.Hd.FibreChannel.Compliance

Fibre Channel devices

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Fibre Channel devices must comply with Fibre Channel Protocol for SCSI, Second Version (FCP-2) or later. To ensure interoperability at the electrical and signaling levels, Fibre Channel devices must comply with Third-Generation Fibre Channel Physical and Signaling Interface (formerly ANSI X3.303:1998).

Send comments about this topic to Microsoft

Device.Storage.Hd.Flush

In this topic:

Device.Storage.Hd.Flush.BasicFunction

Flush command completion

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Industry standard spec requirements:

- For an ATA device, FLUSH CACHE (E7h, Non-Data) 28-bit command is optional for ATA devices and ATAPI devices. FLUSH CACHE EXT (EAh, Non-Data) 48-bit command is mandatory for devices implementing the 48-bit Address feature set.
- For a SCSI device, SYNCHRONIZE CACHE (10) command and/or SYNCHRONIZE CACHE (16) command shall be implemented.

Windows Design Spec requirements - HDD:

- Correct Reporting of Completion: When the OS issues a flush cache command, the storage device should synchronously report the completion of the command only when the content of the cache has been persisted.

Note: The requirement is not applicable to Laptop.

Send comments about this topic to Microsoft

Device.Storage.Hd.Iscsi

In this topic:

Device.Storage.Hd.Iscsi.BasicFunction

iSCSI devices

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

iSCSI devices

iSCSI devices must comply with RFC3720, RFC3721, and RFC3723.

- Device must complete testing by using the Microsoft iSCSI initiator.
- Device must be able to receive ping (ICMP) and send ping (ICMP).
- The following iSCSI protocol features are required:
- Send targets.
- Logon Authentication: CHAP and none. Targets may delegate CHAP authentication to Radius.
- Discovery Session Logon key/value pairs: InitiatorName, SessionType, and AuthMethod.
- Normal Session Logon key/value pairs: InitiatorName, SessionType, AuthMethod, and TargetName.
- DataPDUInOrder.
- DataSequenceInOrder.
- DefaultTime2Wait.
- DefaultTime2Retain
- ErrorRecoveryLevel=0.
- Targets that allow different shared secrets for different initiator names.

The following iSCSI protocol features must pass testing if they are implemented:

- Mutual CHAP.
- HeaderDigest: CRC32 and none.
- DataDigest: CRC32 and none.
- InitialR2T.
- IPsec; when using IPsec, Main mode must be available. In addition, the following items are required when IPsec is implemented:
    - IPsec transport mode must be implemented.

- Internet key exchange (IKE) implementations must support main mode and preshared keys. Target portals with the same IP address must expect the identical main mode IKE policy.
- Targets and initiators must allow different preshared keys for different identifier payloads.
- Targets and initiators must have static IP addresses for main mode.
- Additional Standard Inquiry data VERSION DESCRIPTORS (SPC-3) are required
- At least one iSCSI VERSION DESCRIPTOR is required (value = 0960h).

Send comments about this topic to Microsoft

Device.Storage.Hd.Mpio

In this topic:

Device.Storage.Hd.Mpio.BasicFunction

RAID implementations that provide a multipathing solution must comply with Microsoft multipath I/O (MPIO).

Applies to	Windows Server 2016 Technical Preview x64

Description

Internal or external RAID implementations that provide a multipathing solution must comply with Microsoft multipath I/O (MPIO). Windows multipathing solutions must consist of a Device Specific Module (DSM) created by using the Microsoft MPIO DDK and must comply with all requirements set forth in the Multipath I/O Program Agreement.

Following WMI classes must be implemented by 3rd party DSM.

- DSM_QuerySupportedLBPolicies
- DSM_QueryUniqyeId

3rd party DSM must report minor, major version numbers for the DSM

Send comments about this topic to Microsoft

Device.Storage.Hd.MultipleAccess

Drives that support Multiple Access must meet these requirements.

In this topic:

Device.Storage.Hd.MultipleAccess.MultiplePorts

Multi-port drives must provide symmetric access.

Applies to	Windows Server 2016 Technical Preview x64

Description

Multi-port drives must support the same set of commands on all ports. Drives must not provide different behavior or degraded performance for commands based on which port is used for command delivery.

When drives are connected to a host via multiple paths, the drives must use Windows' Multipath IO (MPIO) solution with the Microsoft Device Specific Module (DSM).

Example: Drives must provide the same performance for data access commands and the same behavior for persistent reservation commands arriving on different ports as they provide when those commands arrive on the same port.

The performance degradation between any two ports should be within a 10% range.

Notes: Multi-port drives may be connected to one or more computer hosts via one or more paths per host. Connecting a drive to multiple hosts enables Windows to use the drive as part of a failover cluster of hosts. Connecting a drive to a single host via multiple paths enables Windows to continue to provide access to the drive in the event of a cable failure. Windows supports using these connection topologies independently and jointly.

Send comments about this topic to Microsoft

Device.Storage.Hd.MultipleAccess.PersistentReservation

Drives that support Persistent Reservations must meet these requirements.

In this topic:

Device.Storage.Hd.MultipleAccess.PersistentReservation.BasicFunction

Drives must provide persistent reservations.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Drives must implement persistent reservations as per the SCSI-3 Primary Commands (SPC-3) specification. Windows depends on proper behavior for the below persistent reservation capabilities.

- PERSISTENT RESERVE IN Read Keys (00h)
- PERSISTENT RESERVE IN Read Reservation (01h)
- PERSISTENT RESERVE OUT Reserve (01h)
  - Scope: LU_SCOPE (0h)
  - Type: Write Exclusive - Registrants Only (5h)
- PERSISTENT RESERVE OUT Release (02h)
- PERSISTENT RESERVE OUT Clear (03h)
- PERSISTENT RESERVE OUT Preempt (04h)
- PERSISTENT RESERVE OUT Register AND Ignore Existing Key (06h)
- PERSISTENT RESERVE OUT Register (00h)

Notes: Windows can use physical disks to form a storage pool. From the storage pool, Windows can define virtual disks called storage spaces. A failover cluster can make the pool of physical disks, the storage spaces they define, and the data they contain highly available. In addition to the standard HCT qualification, physical disks should also pass the "Microsoft Cluster Configuration Validation Wizard" (ClusPrep) tool.

Send comments about this topic to Microsoft

Device.Storage.Hd.OffloadedDataTransfer

Windows Offloaded Data Transfer

In this topic:

Device.Storage.Hd.OffloadedDataTransfer.CopyOffload

If Copy Offload is supported, these requirements must be implemented.

Applies to	Windows Server 2016 Technical Preview x64

Description

Industry standard spec requirements:

- Targets that support the Windows Copy Offload feature must implement the T10 XCOPY Lite specification (11-059r8):
  - Supported VPD pages (Must include ECOP VPD Page (Page Code 8Fh) in the Supported VPD page list)
  - ECOP VPD page or ECOP VPD page (Page Code 8Fh) + Block Device ROD Limits ECOP descriptor (0000h)
  - Block Limit VPD page (Page Code B0h)
  - According to the T10 11-059r8 spec, Windows adopts 83h OP Code + 10 Service Action Code for POPULATE TOKEN and 83h OP Code + 11 Service Action Code for WRITE USING TOKEN commands.
  - According to the T10 11-059r8 spec, Windows adopts 84h OP Code + 07 Service Action Code for the RECEIVE ROD TOKEN INFORMATION command. Response service action field and command parameters shall be compliant with T10 XCOPY Lite spec (11-059r8).

Windows Design Spec requirements:

- During the target device enumeration, Windows will send down an Inquiry for Supported VPD pages VPD page. If 8F is included in Supported VPD page list, Windows will inquire for the ECOP VPD page and BLOCK LIMITs VPD page.
- Implementation and error handling with parameters of the ECOP VPD page
  - The MAXIMUM RANGE DESCRIPTORS - If the number of Block Device Range Descriptors of a POPULATE TOKEN or WRITE USING TOKEN command exceeds the MAXIMUM RANGE DESCRIPTORS, copy manager shall terminate the command with the CHECK CONDITION

status with the sense key set to ILLEGAL REQUEST and the additional sense code set to TOO MANY SEGMENT DESCRIPTORS.

- The MAXIMUM INACTIVITY TIMER (MAXIMUM IAT) - If the INACTIVITY TIMEOUT of a POPULATE TOKEN command exceeds the MAXIMUM INACTIVITY TIMER, copy manager shall terminate the command with the CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.
- The MAXIMUM TOKEN TRANSFER SIZE -
  - If the sum of the NUMBER OF LOGICAL BLOCKS fields in all block device range descriptors of the WRITE USING TOKEN command is greater than the MAXIMUM TOKEN TRANSFER SIZE, copy manager shall terminate the command with the CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.
  - If the sum of the NUMBER OF LOGICAL BLOCKS fields in all block device range descriptors of the POPULATE TOKEN is greater than the MAXIMUM TOKEN TRANSFER SIZE, copy manager shall terminate the command with the CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

- Implementation and error handling with parameters of Block Limits VPD page
- The MAXIMUM TRANSFER LENGTH field indicates the maximum transfer length in blocks that the copy manager accepts for a single BLOCK DEVICE RANGE DESCRIPTOR. If a copy manager receives a request for a NUMBER OF LOGICAL BLOCKS exceeding this maximum, then the copy manager shall terminate the command with CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.
- Storage array must support both synchronous and asynchronous POPULATE TOKEN and WRITE USING TOKEN according to T10 11-059r8, 11-078r4, and 11-204r0 spec.
- Storage array must complete synchronous POPULATE TOKEN and WRITE USING TOKEN commands in a very short time (4 seconds) without causing any SCSI command timeout.

User Experience Requirements:

- Fall back to Legacy Copy - Windows copy offload operation shall be able to fall back to legacy copy operation when a copy offload error or limitation is reported.

- Drag and drop copy experience - must be able support drag and drop copy with copy offload capable storage target device.

Send comments about this topic to Microsoft

Device.Storage.Hd.PersistentReservation

In this topic:

Device.Storage.Hd.PersistentReservation.ClusterFailover

Cluster failover for RAID array systems

Applies to	Windows Server 2016 Technical Preview x64

Description

Required: All Microsoft MPIO device-specific modules (DSMs) must be Windows Hardware Certification-qualified and support registering and unregistering persistent reservations across all paths.

Design Notes:

- All host bus adaptors (HBA) used on failover cluster nodes can use only a Windows Hardware Certification-qualified miniport driver based on the Storport miniport model.
- All multipath I/O solutions leveraged on highly available failover clusters must be based on Microsoft MPIO.
- It is recommended that in addition to the standard HCT qualification all solutions are also validated with the "Microsoft Cluster Configuration Validation Wizard" (ClusPrep) tool.
- FC, iSCSI, and particularly serial-attached SCSI (SAS) failover cluster solutions cannot be built on RAID HBAs where cache and/or RAID configuration is machine/node specific. The RAID set information and hardware cache must reside in a single shared point that lives in an external storage controller.
- SAS, FC, and iSCSI have no restrictions as to the number of nodes they support (which currently is 8nodes).

Note: Legacy parallel-SCSI server clusters were restricted to a maximum size of 2 nodes.

- Only SAS devices using the Serial SCSI Protocol (SSP) transport will be supported on failover clusters (including SAS JBOD or any SAS SSP RAID systems). SATA devices attached to a SAS domain must be part of a RAID system.
- SATA direct attach and SATA JBOD is not supported; the system must include RAID.
- If the system disks are attached to a bus type that is not a valid type for shared storage (something other then FC, iSCSI, or SAS), then the system disks and shared storage must be on separate physical controllers/host bus adaptors.

Send comments about this topic to Microsoft

Device.Storage.Hd.PortAssociation

Drives must provide port association.

In this topic:

Device.Storage.Hd.PortAssociation.BasicFunction

Drives must provide port association.

Applies to	Windows Server 2016 Technical Preview x64

Description

A drive must report its port address for device identification VPD page 83h inquiry association type 1 (port association). This must be the same address the drive supplies to a SCSI Enclosure Services (SES) device and the SES device reports via SES diagnostic page 0Ah with protocol identifier set to 6h.

Notes: Windows depends on drive enclosures to provide SCSI Enclosure Services (SES) capabilities such as drive slot identification and visual drive indications (commonly implemented as drive LEDs). Windows matches a drive in an enclosure with SES identification capabilities via the drive's port address. Computer hosts may be separate from drive enclosures or may be integrated into drive enclosures.

Send comments about this topic to Microsoft

Device.Storage.Hd.RaidArray

In this topic:

Device.Storage.Hd.RaidArray.BasicFunction

RAID array systems

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

RAID Requirements

RAID systems and devices must support commands from SBC-2 (or later) regardless of the drive interface implemented.

RAID controllers and RAID systems must support, at a minimum, one of: RAID1, RAID 5, RAID6, or RAID 1/0.

External RAID arrays must allow a failed drive that is redundant to be replaced manually without shutting down or halting the system. This requirement includes, but is not limited to, drives in a mirror set, a physical drive being replaced by a "hot spare," and the first failed drive in a RAID level-5 array. The RAID subsystem must also allow lost data to be rebuilt without interfering with system operations. It is expected that RAID array throughput will be impacted during the rebuild.

Device.Storage.Hd.RaidArray.BitLocker

BitLocker must not cause data corruption on storage arrays.

Applies to	Windows Server 2016 Technical Preview x64

Description

BitLocker must be properly enabled to protect data volumes on storage arrays.

Business Justification

(1) When a server is placed in an environment without adequate physical security, BitLocker protects data on the server against unauthorized access if a server is stolen. (2) When hosting service providers repurpose or decommission storage arrays, BitLocker Disk Encryption prevents data breach.

Device.Storage.Hd.RaidArray.Manageability

A connected storage device must be manageable.

Applies to	Windows Server 2016 Technical Preview x64

Description:

A storage device is considered connected when it operates with a Windows Server system through a fabric interface. This type of storage interface includes FibreChannel, FibreChannel over Ethernet (FCoE), iSCSI, and SMB3.

This manageability of a connected storage device means the user can perform an end-to-end storage workflow using a common user interface such as PowerShell, or the File and Storage Services in a Server Management tool. This workflow must be achievable even before the operational target Windows Server OS is deployed in the environment.

Enforcement

This is a mandatory device requirement. This requirement becomes in effect at the release of Windows Server vNext.

Business Justification

Manageable connected storage devices allow efficient deployment and operation of server systems running Windows Server vNext in a software-defined datacenter and therefore lowering operational costs for the customers where heterogeneous hardware platforms are deployed.

Device.Storage.Hd.RaidArray.Manageability.Smapi

A connected storage device must be manageable by Windows through its Storage Management API (SMAPI) and an associated Storage Management Provider (SMP).

Applies to	Windows Server 2016 Technical Preview x64

Description:

A storage device is considered connected when it operates with a Windows Server system through a fabric interface. This type of storage interface includes FibreChannel, FibreChannel over Ethernet (FCoE), iSCSI, and SMB3.

This manageability of a connected storage device means the user can perform an end-to-end storage workflow using a common user interface such as PowerShell, or the File and Storage Services in a

Server Management tool. This workflow must be achievable from a Windows Server vNext, through its Storage Management API (SMAPI) and an associated Storage Management Provider (SMP), to the connected storage device providing the services.

Enforcement

This is a mandatory device requirement. This requirement becomes in effect at the release of Windows Server vNext.

Business Justification

Manageable connected storage devices allow efficient deployment and operation of server systems running Windows Server vNext in a software-defined datacenter and therefore lowering operational costs for the customers where heterogeneous hardware platforms are deployed.

Device.Storage.Hd.RaidArray.Manageability.Smi

An industry-standard connected storage device is manageable through its native SNIA SMI-S v1.6.1 Provider interface.

Applies to	Windows Server 2016 Technical Preview x64

Description:

A storage device is considered connected when it operates with a Windows Server system through a fabric interface. This type of storage interface includes FibreChannel, FibreChannel over Ethernet (FCoE), iSCSI, and SMB3.

A connected storage device is considered industry-standard when it supports the Storage Networking Industry Association (SNIA) Storage Management Initiative specification (SMI-S) v1.6.1 for manageability purposes. This SMI-S v1.6.1 Provider interface must be implemented in the connected storage device without the aid of an external companion provider operating in a separate device enclosure.

This manageability of an industry-standard connected storage device means the user can perform an end-to-end storage workflow using a common user interface such as Windows PowerShell, or the file and storage services in a server management tool such as Microsoft System Center Virtual Machine Manager (SCVMM). This workflow must be achievable even when a Windows Server OS is not yet deployed in the environment.

Enforcement

This is an “If-Implemented” optional device requirement for a connected storage device.

However, this and all related sub-requirements are mandatory for a connected storage device claiming to be industry-standard and manageable from a Windows Server vNext system. This requirement becomes in effect at the release of Windows Server vNext.

Business Justification

Manageable connected storage devices allow efficient deployment and operation of server systems running Windows Server vNext in a software-defined datacenter and therefore lowering operational costs for the customers where heterogeneous hardware platforms are deployed.

Device.Storage.Hd.RaidArray.Manageability.Smi.Ctp

An industry-standard connected storage device that is manageable through its native SMI-S v1.6.1 Provider interface demonstrates such conformance through a successful CTP test pass.

Applies to	Windows Server 2016 Technical Preview x64

Description:

A storage device is considered connected when it operates with a Windows Server system through a fabric interface. This type of storage interface includes FibreChannel, FibreChannel over Ethernet (FCoE), iSCSI, and SMB3.

A connected storage device is considered industry-standard when it supports the Storage Networking Industry Association (SNIA) Storage Management Initiative specification (SMI-S) v1.6.1 for manageability purposes. This SMI-S v1.6.1 Provider interface must demonstrate such conformance through a successful test pass of the SNIA SMI-S Conformance Testing Program (CTP), and the test result must be endorsed by the SNIA SMI Lab Conformance Committee.

Enforcement

This is a mandatory device requirement for a connected storage device claiming to be industry-standard and manageable from a Windows Server vNext system. This requirement becomes in effect at the release of Windows Server vNext.

Business Justification

Manageable connected storage devices allow efficient deployment and operation of server systems running Windows Server vNext in a software-defined datacenter and therefore lowering operational costs for the customers where heterogeneous hardware platforms are deployed.

Device.Storage.Hd.RaidArray.Manageability.Smi.Scvmm

An industry-standard connected storage device that is manageable through its native SMI-S v1.6.1 Provider interface demonstrates its interoperability with the Microsoft SCVMM vNext.

Applies to	Windows Server 2016 Technical Preview x64

Description:

A storage device is considered connected when it operates with a Windows Server system through a fabric interface. This type of storage interface includes FibreChannel, FibreChannel over Ethernet (FCoE), iSCSI, and SMB3.

A connected storage device is considered industry-standard when it supports the Storage Networking Industry Association (SNIA) Storage Management Initiative specification (SMI-S) v1.6.1 for manageability purposes. This SMI-S v1.6.1 Provider interface must demonstrate interoperability with the Microsoft System Center Virtual Machine Manager (SCVMM) vNext running on a Windows Server vNext system.

This interoperability with SCVMM vNext demonstrates the connected storage device manageability through the Windows Storage Management API (SMAPI) and a combination of the Windows Standards-Based Storage Management Service and its native SMI-S v1.6.1 Provider. This manageability means the user can perform an end-to-end storage workflow using a common interface such as PowerShell, or the File and Storage Services in SCVMM.

Enforcement

This is a mandatory device requirement for a connected storage device claiming to be industry-standard and manageable from a Windows Server vNext system. This requirement becomes in effect at the release of Windows Server vNext.

Business Justification

Manageable connected storage devices allow efficient deployment and operation of server systems running Windows Server vNext in a software-defined datacenter and therefore lowering operational costs for the customers where heterogeneous hardware platforms are deployed.

Send comments about this topic to Microsoft

Device.Storage.Hd.ReadZeroOnTrimUnmap

In this topic:

Device.Storage.Hd.ReadZeroOnTrimUnmap.BasicFunction

The requirement applies to hard disk drives.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the logical block provisioning read zeros (LBPRZ) bit is set to one, then the device server shall set all bits to zero in the Data-In Buffer for a read operation on an unmapped (deallocated or anchored) LBA.

Send comments about this topic to Microsoft

Device.Storage.Hd.RemovableMedia

Defines requirements that must be met if the storage device is removable, i.e., it has RMB bit set to 1.

In this topic:

Device.Storage.Hd.RemovableMedia.BasicFunction

Devices with true removable storage media

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Device with true removable storage media should report as True Removable Media (RMB=1) according to SPC-4 Revision 29, Section 6.4.2.

Send comments about this topic to Microsoft

Device.Storage.Hd.Sas

In this topic:

Device.Storage.Hd.Sas.ComplyWithIndustrySpec

Serial attached SCSI devices must comply with industry specifications.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The reference for specification compliance. Where noted as Min, the baseline specification is mandatory. “Rec:” indicates the preferred version of the specification. If not otherwise specified, the

version listed is the minimum required. Unless otherwise indicated, all features of the cited specifications that are classified as mandatory by the standards body must be implemented.

SAS-1, SAM-3, SPC-3, Min:SBC-2, Rec: SBC-3

Serial Attached SCSI devices comply with the Serial Attached SCSI (SAS) Specification 1 or later.

- Shall not cause more than 10% IO performance degradation when the SAS storage device is used in a MPIO configuration.
- SAS storage device shall establish a UNIT ATTENTION subsequent to detection of the following events.
    - Power On
    - Hard Reset
    - Logical unit reset
    - I_T Nexus loss
    - Power loss expected
- SAS SSD must implement following T10 (SCSI) command specification:
    - Read Capacity (16)
    - Block Limit VPD Page (Page Code B0h)
    - Block Device Characteristics VPD page (Page Code B1h)
    - Logical Block Provisioning VPD Page (Page Code B2h)
        - SAS devices must support the Task Abort functionality described in the SAM-5 spec (T10).
- SAS SSD must meet the following requirements:
    - SAS SSD target device must return MEDIUM ROTATION RATE = 0001h (Non-rotating medium)
    - SAS SSD target device must return Read Capacity (16) command with LBPME bit set to 1 and Provisioning Type field = 0 (000b) Not Report a Provisioning Type or 1 (001b) Resource Provisioned in the Logical Block Provisioning VPD page (Page Code B2).
    - SAS SSD device must implement Block Limit VPD Page (Page Code B0h) and support the following parameters.

        - MAXIMUM UNMAP LBA COUNT
        - MAXIMUM UNMAP BLOCK DESCRIPTOR COUNT
        - OPTIMAL UNMAP GRANULARITY
        - UNMAP GRANULARITY ALIGNMENT
        - UGAVALID Bit
    - SAS SSD target device must implement Logical Block Provisioning VPD Page (Page Code B2h) and support the following parameters:
        - LBPU bit
        - LBPRZ bit
        - Provisioning Type field.
    - SAS SSD must support UNMAP (10) command and the LBPU bit in LBP VPD page shall set to one.
        - If the device reports a seek penalty (SMR), the following thresholds apply:
            - 98.5% of all I/O must complete within 200ms
            - 100% of all I/O must complete within 1,000ms
        - Otherwise (SSD):
            - 98.5% of all I/O must complete within 100ms
            - 100% of all I/O must complete within 500ms
- If the LBPME bit in ReadCapacity(16) return is set to one, the SAS SSD device must support Logical Block Provisioning VPD page (Page Code B2h)
- If the LBPRZ bit in ReadCapacity(16) return is set to one, the SAS SSD device must set LBPRZ bit of Logical Block Provisioning VPD page to one.

[If Implemented] Firmware Download & Activate

SAS devices implementing the ability to download and activate firmware, i.e. command WRITE BUFFER, shall behave in the following way:

- Mode 0Eh (Download Microcode with offsets, save, and defer activate) and 0Fh (Activate deferred microcode) of the command WRITE BUFFER shall be supported by the device.

- The device must retain Persistent Reservations through the firmware download/activate cycle.
- The device must retain VPD page 0x83 through download and activate of a firmware image.
- The download operation must be possible using SCSI Buffer ID 0 and a transfer size of 64KB.
- Execution of the WRITE BUFFER command (subcommands 0Eh/0Fh) shall not affect the execution of other commands in the device queue. If I/O is failed– it must fail with sense code MICROCODE HAS BEEN CHANGED.
- The activation process must occur without requiring the host to power cycle or bus reset, i.e. activation must be completed via an internal reset of the SAS drive.

Send comments about this topic to Microsoft

Device.Storage.Hd.Sata

In this topic:

Device.Storage.Hd.Sata.BasicFunction

ATA device performance

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

SATA Devices:

- Requirement: SATA devices shall meet the requirements of the Serial ATA: High Speed Serialized AT Attachment, Version 2.6 or later.
- Recommendation: SATA devices should implement interface power management
- Recommendation: SATA devices should implement Native Command Queuing (NCQ) support
- Hot-Plug:

- If the device is connected to a port marked as “external” hot-plug support is required. A port is considered “external” if any one of the following register settings (according to the AHCI industry spec) is true:
  - PxCMD.HPCP = 1 or,
  - CAP.SXS = 1 and PxCMD.ESP = 1 or,
  - CAP.SMPS = 1 and PxCMD.MPSP = 1
- If the device is connected to a port marked as “internal” no interrupt needs to be generated for a hot-plug event

[If Implemented] Firmware Download & Activate

SATA drives implementing the ability to download and activate firmware, i.e. command DOWNLOAD MICROCODE DMA (IDENTIFY DEVICE word 69 bit 8 == 1) or command DOWNLOAD MICROCODE (IDENTIFY DEVICE word 83 bit 0 == 1) shall behave in the following way:

- The device must retain the following IDENTIFY device data through download and activate of a firmware image: Vendor ID, Model ID, Serial Number.
- The activation process must occur without requiring the host to power cycle or bus reset, i.e. activation must be completed via an internal reset of the SATA device.
- [DMA] The drive shall support IDENTIFY DEVICE data log page 03h.
  - The field “DM OFFSETS DEFERRED SUPPORTED bit” shall be set to 1, indicating that subcommands 0EH and 0FH are supported.
- [DMA] Subcommand 0Eh “Download with offsets and save microcode for future use” of the command DOWNLOAD MICROCODE DMA shall be supported.
- [DMA] Subcommand 0Fh “Activate downloaded microcode” of the command DOWNLOAD MICROCODE DMA shall be supported.

Note: A DMA implementation of these commands is preferred but not required.

Send comments about this topic to Microsoft

Device.Storage.Hd.Sata.HybridInformation

This feature is for all devices that support the Hybrid Information feature.

In this topic:

Device.Storage.Hd.Sata.HybridInformation.BasicFunction

SATAIO hybrid drive requirements

Applies to	Windows 10 x64 Windows 10 x86

Description

The following functionalities and requirements have to be supported by the device and refer to hybrid devices in general. They are required for use with the Windows in-box driver (StorAHCI.sys).

If a drive reports itself as a hybrid drive, it must be comprised of a spinning disk and at least one non-volatile cache (NVC) component within a single physical device.

Self-Pinning on Boot

The device shall implement a mechanism to self-pin boot files, i.e. anything prior to Windows becoming active, into the top priority level of the cache. Specifically, the drive shall self-pin in this fashion from power on until the Hybrid Information Log Page is read. The drive should cease self-pinning to the top priority after 160 MB of data has been pinned. All other self-pinning the drive performs shall occur at priority 0.

Cache Size

The hybrid cache has to provide at least 5 GB (5x230 Bytes) of useable capacity to the host.

Note: Devices providing less than 12 GB of useable cache capacity will not be eligible to benefit from hiberfile, swapfile, or other sequential I/O caching.

6 priority levels (0 through 5)

The device must support at least 6 priority levels. No data placed in the cache by the host shall be evicted from ranges 1 through 5 by the device, unless Windows specifies it through calls to evict, demotebysize or trim, or issues further I/Os with priority > 0 to a full cache (i.e. cache churning).

Priority 0

Any I/O hinted at priority 0 shall be serviced directly by the platter, regardless of current state (i.e. if the platter is spun down and receives a priority 0 I/O, it shall spin-up). I/O hinted at priority 0 shall bypass the non-volatile cache and invalidate any cache-resident LBAs that are hinted at priority 0 and now present on platter.

Direct Flash Access

When the platter is spun down, all hinted write commands – except priority 0 writes – shall be serviced directly by the non-volatile cache. The write I/O shall not be staged on the platter first, so as to avoid platter spin-up. If the write cannot be serviced by the non-volatile cache, due to resource constraints (no clean pages, etc.) the platter may spin-up to service the I/O.

Power Up In Standby

The device must support the Power Up In Standby feature (PUIS) defined in SATA spec ATA ACS-3.

Cache Page Replacement

The drive shall repurpose cache pages of lower priorities to satisfy I/Os at higher priorities, starting with the lowest priority. For example, if the cache is full with 25% at priority 0, 50% at priority 2, and 25% at priority 3 and a priority 3 read/write occurs that is not in cache, a priority 0 cache page should be repurposed to accommodate the priority 3 LBA that was read. Once all priority 0 cache pages are repurposed, priority 1 and 2 pages will be repurposed respectively. If no lower priority pages are available, priority 3 should repurpose pages first that are chosen by a LRU-like algorithm.

Update Priority On IO (Read/Write)

Priorities associated with LBAs in the cache must be updated on any subsequent read or write to that LBA, from any valid priority to any valid priority. In addition, if a read or write is issued to an LBA range that currently does NOT reside in cache, then it shall be placed in the cache at the specified priority (provided there is room in the cache or there are blocks of lower priority that can be evicted – in case of a full cache).

Trim

The device shall support trim as defined in the Windows 8 Hardware Certification Requirements under Device.Storage.Hd.Trim.BasicFunction.

Non-volatile Cache Performance:

Random Read:>= 8 MB/s @ 4 KB (Queue Depth = 1)

Latency:

Setting dirty high and dirty low thresholds shall complete within 50 ms.

This command can be completed asynchronously.

I/O that carries priority information should not be significantly slower than I/O without priority information. A read or write that has a priority assigned must not incur a latency penalty larger than the bigger of 10% of an identical I/O without priority information or 0.5 ms (max(10%,0.5 ms)). Achieving this on 95% of all I/Os is acceptable with the penalty never exceeding 100% of a non-priority I/O.

HybridDemoteBySize must return within 500ms when called on 3/255th of the cache size.

HybridChangePriorityByLBARange must return within 150 ms when called on a 32 MB range of data.

A possible asynchronous implementation of these commands is acceptable.

Command/Protocol Requirements

The following functionalities and requirements have to be supported by the device and refer SATA revision 3.2 worked on by SATAIO:

13.7.5.4.11 MaxPriority Behavior

The MaxPriority Behavior bit must be set to 0.

13.3.15 Enable/Disable Hybrid Information

13.6.5.4.13 Hybrid Control

Enable/Disable Caching Medium

Windows must have the ability to enable and disable the hybrid cache. Upon disabling the hybrid cache, all dirty data in the cache has to be synchronized with the final storage medium.

Dirty Low Threshold

Dirty High Threshold

Dirty data should be synchronized in ascending order of priority, i.e. priority 0 dirty data should be synchronized before priority 1 dirty data is synchronized. When the amount of dirty data on the device, exceeds the dirty high threshold marker, the device must begin the synchronization of dirty data. It is understood that this requires the platter to spin-up.

13.6.5.4.7 Hybrid Demote By Size

Hybrid Information feature related Logs

The device must enable Windows to read the hybrid log pages and return sensible information. In other words, General Purpose Logging is required and the version number shall be set to 0001h. Specifically:

13.7.9 Word 0x12 of “General Purpose Log Directory” shall be set to “1”, indicating that the “NCQ NON-DATA” log is supported.

13.7.10 Word 0x13 of “General Purpose Log Directory” shall be set to “1”, indicating that the “NCQ SEND AND RECEIVE” log is supported.

13.7.12 Word 0x14 of “General Purpose Log Directory” shall be set to “1”, indicating that the “NCQ HYBRID INFORMATION” log is supported. Windows has to be able to determine the following:

Caching Medium Health Status

Dirty Low Threshold

Dirty High Threshold

Maximum Hybrid Priority Level

Max Priority Behavior

NVM size

Max Eviction Commands.

Max Eviction Data Blocks

For each Priority level:

consumed NVM size fraction

consumed Mapping Resources Fraction

consumed NVM Size for Dirty Data Fraction

consumed Mapping Resources for dirty data fraction

13.6.5.4.1 Hybrid Evict

The LBA range specified by the Evict command shall be targeted to the primary medium like a regular write. The host will ensure consistency – if necessary – by issuing a subsequent flush command. Evicted LBA ranges should be invalidated in the NVC, such that the data only resides on the platter after completion of the Evict command.

13.6.5.4.10 Hybrid Change By LBA Range

Hybrid Change By LBA Range must be supported by the device, including the Cache Behavior bit of the command.

Send comments about this topic to Microsoft

Device.Storage.Hd.Scsi

In this topic:

Device.Storage.Hd.Scsi.Connectors

SCSI connectors

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

SCSI connectors

If an external connector is implemented, it must meet the requirements in SCSI or a later specification. The SCSI connector must not use the same connector type as any other non-SCSI connector on the system. All external parallel SCSI connectors must be labeled with an ANSI-approved icon for the bus. For internal and external configurations, the SCSI bus cable must be plugged into shrouded and keyed connectors on the host adapter and devices. This ensures that the cable is properly positioned so the user cannot plug in cables incorrectly. For internal configurations, pin-1 orientation must be designated on one edge of the ribbon cable and also on the keyed connector for the SCSI peripheral device.

Device.Storage.Hd.Scsi.ParallelInterface

Parallel SCSI interface

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Parallel SCSI interface

- Parallel SCSI devices and adapters comply with SCSI Parallel Interface-4 (SPI-4) or later.
- Termination: Automatic termination circuit and SCSI terminators meet SPI-4 standard or later. Parallel SCSI host controllers and adapters must use automatic termination that allows a user to add external devices without removing the server case. Terminators used in the SCSI host adapter must be regulated terminators, which are also known as active, SCSI SPI-4, or Boulay terminators. SCSI termination built onto internal cables must also meet the SPI-4 specification.
- Terminator power must be supplied to the SCSI bus with overcurrent protection. The host adapter must supply terminator power (TERMPWR) to the SCSI bus for system-board implementations by using PCI or another expansion bus. All terminators on the external SCSI bus must be powered from the TERMPWR lines in the SCSI bus. In addition, the circuit that supplies TERMPWR must have overcurrent protection built into it.
- External removable disks, hard drives, and CD/DVD optical drives must provide automatic termination or an accessible on-board termination switch. At a minimum, a mechanical means must be provided for setting termination and the switch must be accessible to the user without opening the device chassis.
- All SCSI devices supporting hot plugging must comply with annex D of SPI-4, which addresses SCSI device insertion and removal, with and without command activity.
- Differential devices must support DIFFSENS as defined in SPI-4 standard or later.

Send comments about this topic to Microsoft

Device.Storage.Hd.Scsi.ReliabilityCounters

Basic reliability counter functionality for disks that implement the SCSI command sets.

In this topic:

Device.Storage.Hd.Scsi.ReliabilityCounters.BasicFunction

Basic reliability counter functionality for disks that implement the SCSI command sets.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All SCSI drives must provide valid data for the below log sense page (LOG SENSE 4Dh) parameters as per the SCSI Primary Commands 4 (SPC-4) and SCSI Block Commands 3 (SBC-3) specifications.

- Start-Stop Cycle Counter (0Eh)
  - Manufacture Date (0001h)
- Read Error Counter (03h)
  - Total (0002h)
  - Total Errors Corrected (0003h)
  - Total Uncorrected Errors (0006h)
- Temperature (0Dh)
  - Temperature (0000h)
  - Reference Temperature (0001h)
- Write Error Counter (02h)
  - Total (0002h)
  - Total Errors Corrected (0003h)
  - Total Uncorrected Errors (0006h)
- Background Scan (15h)
  - Background Scan Status (0000h)

Drives which physically move recording media and/or read-write devices, such as hard disk drives, must provide valid data for the below log sense page (LOG SENSE 4Dh) parameters as per the SCSI Primary Commands 4 (SPC-4) specification.

- Start-Stop Cycle Counter (0Eh)

- Specified Cycle Count Over Device Lifetime (0003h)
- Accumulated Start-Stop Cycles (0004h)
- Specified Load-Unload Count Over Device Lifetime (0005h)
- Accumulated Load-Unload Cycles (0006h)

Solid-state drives must provide valid data for the below log sense page (LOG SENSE 4Dh) parameter as per the SCSI Block Commands 3 (SBC-3) specification.

- Solid State Media (11h)
  - Percentage Used Endurance Indicator (0001h)

Send comments about this topic to Microsoft

Device.Storage.Hd.ScsiProtocol

In this topic:

Device.Storage.Hd.ScsiProtocol.ReferenceSpec

Reference to specifications

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Where noted as Min, the baseline specification is mandatory. Rec indicates the preferred version of the specification. If not otherwise specified, the version listed is the minimum required. Unless otherwise indicated, all features of the cited specifications that are classified as mandatory by the standards body must be implemented.

SPI-4, SAM-3, Min:SPC-2, Rec: SPC-3, Min: SBC, Rec: SBC-2

Device.Storage.Hd.ScsiProtocol.SamCompliance

SCSI Architecture Model SAM-3

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

SCSI Architecture Model SAM-3

SCSI Devices must comply with SCSI Architecture Model SAM-3 or later (except as noted in SBP-2 for 1394 devices), including the following requirements:

- All devices must support LUN reset. In particular, if two LUNs L0 and L1 under the same target have outstanding commands, a LUN reset to L0 must clear any outstanding commands to L0 only.
- Following a reset, all devices must return an appropriate unit attention condition to any initiator currently having access to the logical unit.
- All FC, iSCSI, SCSI, and SAS devices must support multiple initiators.
- MODE SELECT commands that change parameters must cause a unit attention condition to be raised for any other initiator consistent with SAM-3.
- LUN 0 must be implemented for all targets. At a minimum, LUN 0 must respond to INQUIRY and all multi-LUN targets must support the REPORT LUNS commands.
- If any LUN is added or removed that is accessible to the initiator(s), the device must report a unit attention condition of (06/3F/0E) REPORT LUNS DATA HAS CHANGED MODE SELECT. Commands that change parameters must cause a unit attention condition to be raised for any other initiator that would be impacted by the change.
- Any unrecognized SCSI command or incorrectly formed command descriptor block (CDB) must result in an immediate CHECK CONDITION reported back to the initiator.

Device.Storage.Hd.ScsiProtocol.SpcCompliance

SCSI Primary Commands (SPC-3, SPC-4 and SBC-3)

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

SCSI Primary Commands-3 (SPC-3), SCSI Primary Commands-4 (SPC-4) and SCSI Block Commands-3 (SBC-3)

Devices must comply with SCSI Primary Commands: support commands listed as mandatory in the SCSI Primary Commands (SPC-3 or later). In addition, each device type must implement the mandatory command set for that type (SBC-3 for block devices and so on).

For SCSI INQUIRY and REPORT LUNS commands:

All devices must support the SCSI INQUIRY command.

Multi-LUN devices must always respond to an INQUIRY command sent to LUN 0 even if LUN 0 is not implemented. This can be indicated by returning the Device Type Qualifier of 3.

- All multi-LUN devices must support the REPORT LUNS command as defined in SPC-2 or later.
- Windows supports only single-level logical unit numbers up to 255; see SAM-3. Use of any other format will be incorrectly interpreted, and the device may not be available or data corruption will occur.

All standard INQUIRY data must be correctly set for the device capabilities.

- VERSION Field must be 04 or greater. For SAS, this field must be 05 or greater.
- Drives with attached media changers must set the MChgnr bit in the standard inquiry data.
- Multi-LUN units must return valid REPORT LUNS data for LUN 0.
- If LUN 0 is not an accessible PERIPHERAL DEVICE TYPE, the PERIPHERAL QUALIFIER shall be returned as 1. SCSIport will not enumerate the entire target device if a qualifier of 3 is used. It is strongly recommended that LUN 0 not be type 0 because it must be exposed to all initiators. Type 0 is permitted only if the array can map different logical units to LUN 0 for each initiator.
- If a device has more than one port, MultiP bit must be set and page 83h descriptors must correctly reflect the port information.

Vital Product Data (VPD) pages:

- Page 00h (Supported Pages VPD Page) is required.
- Page B1h (Block Device Characteristics VPD Page) is required.
- Page 80h (Unit Serial Number VPD Page) is required.

Page 83h (Device Identification VPD Page) is required. For VPD Page 83, at least one type-3 or one type-2 descriptor must be returned for each logical unit, the value must use Code Set 1 (Binary), the value must be unique for that logical unit, and it must be the same value regardless of the path or port responding to the request. Appropriate descriptors for multiport devices are required in addition to that mandatory descriptor. Devices that support aliases must also support the corresponding descriptor types. Vendor-specific device identifiers, if present, must use type 0 and must follow the specified format, including correct page length.

- Vendor-specific identifiers are not a substitute for the mandatory type 2/3 descriptors. All device identifiers must conform to formatting rules set forth in SPC-3 or later, even if the device claims only conformance to a previous release.
- Device must comply with SPC-3 section 7.6.3 Device Identification VPD page 83h.

At least one identification descriptor must have the IDENTIFIER TYPE field set to:

- 2h (EUI-64-based) as defined in 7.6.3.5
- 3h (NAA); or as defined in 7.6.3.6
- 8h (SCSI name string) at defined in 7.6.3.11

SCSI Mode Sense Command and Pages

MODE SENSE (6) is mandatory for all devices except RBC devices, which implement MODE SENSE (10). The DBD bit must be supported.

- Mode Page 3Fh (Return all pages mode page) is mandatory.
- Device type-specific pages listed in the device-specific sections of this document.
- Additional commands for all devices are as follows:
- All devices must support the TEST UNIT READY and REQUEST SENSE commands.

Block Storage (Disk and RAID) Devices

Block storage (disk and RAID) devices must comply with the following requirements:

- SCSI block commands (SBC) or later (RBC for 1394). These requirements apply to any device reporting as Device Type 0, including logical units exposed by a RAID controller or subsystem.
- Block Devices must support the SCSI START STOP UNIT command to decrease power consumption.

- READ CAPACITY (10) command. If a device has more than 2^32 - 1 sectors, a value of 0xFFFFFFFF must be returned for the RETURNED LOGICAL BLOCK ADDRESS field and the READ CAPACITY (16) command must be supported (see below).
- Any change to capacity must set a unit attention condition of CAPACITY DATA HAS CHANGED for all initiators with access to the logical unit.
- READ(10).
- WRITE(10). Support for force unit access (FUA) is mandatory for individual physical disk drives or RAID controllers that contain volatile (non-battery-backed) cache memory and must cause the data sent with this command to be committed to physical media before the command completes.
- REASSIGN BLOCKS (hard disks only). RAID controllers that handle bad block replacement should succeed this command.
- VERIFY (10).
- START STOP UNIT. This command must not perform any other action, such as path failover.
- SYNCHRONIZE CACHE (10) (no optional fields are used). For a physical disk drive, this command causes all data in the write cache to commit to physical media if write caching is enabled. Failure to follow this can result in data corruption.

Mode pages:

- Mode Page 08h (Caching mode page) with the following bits must contain valid information: WCE (Write Cache Enable), CACHE SEGMENT SIZE, and NUMBER OF CACHE SEGMENTS (optional). RBC devices support Page 6 instead of Page 8 (WCD, WRITED, FORMATD, and LOCKD bits).
    - If a device supports disabling write caching through the use of the WCE bit, this bit must also be reported as changeable and be supported by a MODE SELECT operation which modifies it. The status of the write caching must be visible by reading Mode Page 8. Vendors can implement caching policies outside of the limited SBC ones, and disabling of write cache does not need to be through this mode page.
- Mode Page 0Ah (Control mode page) is required.
- Mode Page 1Ah (Power Condition mode page) is required

Disk devices that support greater than 2-TB logical units (including 1394 disks). Devices must conform to SPC-3 and must implement all of the following in accordance with the SCSI Block Commands-2 (SBC-2) specification:

- READ CAPACITY (16)
- READ (16)
- WRITE (16) FUA (bit must be supported if a volatile cache is present on the device)
- VERIFY (16)
- REASSIGN BLOCKS. LONGLBA field must be supported.

Erasable SCSI Disk Devices

Erasable SCSI disk devices must also support the following commands or features:

- ERASE: Full-side and selected-block erase.
- Format requirements reported with FORMAT command.
- MODE SENSE (6) Total spare blocks available, write protect status.
- PREVENT ALLOW MEDIUM REMOVAL and START STOP UNIT.
- REASSIGN BLOCKS and READ DEFECT DATA (10).

WRITE without pre-erase, for erasable optical only.

Send comments about this topic to Microsoft

Device.Storage.Hd.ThinProvisioning

In this topic:

Device.Storage.Hd.ThinProvisioning.BasicFunction

Thin Provisioning

Applies to	Windows Server 2016 Technical Preview x64

Description

Industry standard spec requirements:

- Targets that support thin provisioning feature must implement following T10 SPC4 and SBC3 specification:
- Supported VPD Page VPD Page (Page Code 00h)
- Block Limit VPD Page (Page Code B0h)
- Logical Block Provisioning VPD Page (Page Code B2h)
- Logical Block Provisioning Log Page (Page Code 0Ch)

Windows Design Spec requirements:

- Target devices with thin provisioning feature must meet the following requirements.
  - Must return Inquiry command for Supported VPD Page VPD Page with B0h and B2h.
  - Must return LBPU bit set to one and Provisioning Type field = 3 (010b) of Logical Block Provisioning VPD page (Page Code B2).
  - Must implement Block Limit VPD Page (Page Code B0h) and support the following parameters.
    - MAXIMUM UNMAP LBA COUNT
    - MAXIMUM UNMAP BLOCK DESCRIPTOR COUNT
    - OPTIMAL UNMAP GRANULARITY
    - UNMAP GRANULARITY ALIGNMENT
    - UGAVALID Bit
  - Must implement Logical Block Provisioning VPD Page (Page Code B2h) and support the following parameters.
    - Threshold Exponent
    - LBPU bit
    - LBPRZ bit
    - Provisioning Type field

- Thin Provisioning target devices should support Log sense command to retrieve Logical Block Provisioning Log Page (Page Code 0Ch) - for adding the following information into the threshold notification system event log.
  - Used LBA mapping resources of a Thin Provisioning LUN.
  - Available LBA mapping resources to the Thin Provisioning LUN.
- Storage array must support UNMAP (10) command, the LBPU bit in LBP VPD page shall set to one.
- Must support Get LBA Status (16) command according to T10 SBC3 spec.

- If the LBPME bit in ReadCapacity(16) return is set to one or B2h is reported in the Supported VPD Page VPD Page, the storage array must support Logical Block Provisioning VPD page (Page Code B2h).
- If the LBPRZ bit in ReadCapacity(16) return is set to one, the storage array must set LBPRZ bit of Logical Block Provisioning VPD page to one.
- Storage array must support threshold notification (TN), temporary resource exhaustion (TRE) and permanent resource exhaustion (PRE) through the following sense key, additional sense code and additional sense code qualifier returns as SPC4 and SBC3 specs.
  - TN- Sense Key/ASC/ASCQ (06/38/07)
  - TRE- Sense Key/ASC/ASCQ (02/04/14)
  - PRE- Sense Key/ASC/ASCQ (07/27/07)

User Experience Requirements:

- Must be able to set threshold through vendors' storage management utility and monitor the system event log when the thin provisioning soft threshold is reached.
- Must support Log Sense command to retrieve the LBP log page for reporting available LBA mapping resource and used LBA mapping resource information to the thin provisioning LUN, if Log Page (Page Code OCh) is implemented.

Send comments about this topic to Microsoft

Device.Storage.Hd.Trim

In this topic:

Device.Storage.Hd.Trim.BasicFunction

ATA Trim and SCSI Unmap functionality

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the device implements ATA non-NCQ Trim or SCSI Unmap support:

- The Trim implementation shall comply with ATA ACS2 Section 7.10 (Data Set Management Commands).
- The SCSI Unmap command implementation shall comply with T10 SBC3 Section 5.28 (UNMAP command).
- If the device reports a seek penalty indicating rotational media:
  - All IO and Trim/Unmap commands shall be completed in less than 1,000 ms.
  - 98.5% of IO commands shall be completed in less than 200 ms.
- Otherwise (SSD):
  - All IO and Trim/Unmap commands shall be completed in less than 500 ms.
  - 98.5% of IO commands shall be completed in less than 100 ms.

  If the RZAT bit is set on a SATA device or the LBPRZ bit is set on a SCSI device, then the device shall return all '0's to a Read command before the trimmed block(s) is(are) re-written.

Send comments about this topic to Microsoft

Device.Storage.Hd.Uas

In this topic:

Device.Storage.Hd.Uas.Compliance

USB UAS storage devices

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB UASP storage devices must also be compliant with the USB UASP v1.0 and SPC4, SBC3.

USB UASP Storage devices must support the following:

- Mode page code: 0x08 Mode subpage code : 00
- Block Limits page - 0xB0 SPC3 6.5.3
- Support at least 16 streams
- Support task management commands

Note: For further information on mode pages, see SPC4: D.6 Mode page codes.

- Support SPC, SBC version descriptors
- Support/report R02. R04 version descriptors
- Device must report FIXED if it is not a true removable media (RMB=0)

Note: For further information on mode pages, see SPC4: D.6 Mode page codes.

Data devices must perform as indicated:

- Minimum sequential write speed: 100 MB/s
- Minimum sequential read speed: 110 MB/s

Additional Requirement: If the device supports UASP on XHCI, then it must support UASP on EHCI.

Send comments about this topic to Microsoft

Device.Storage.Hd.UasOnEHCI

In this topic:

Device.Storage.Hd.UasOnEHCI.BasicFunction

USB UAS storage devices

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the device supports UASP on XHCI and then it must support UASP on EHCI.

Send comments about this topic to Microsoft

Device.Storage.Hd.Usb

In this topic:

Device.Storage.Hd.Usb.Compatibility

USB compatibility

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB compatibility

- All USB storage devices must meet the requirements of the Universal Serial Bus Mass Storage Class Specification Overview, V1.2 Revision. This includes all USB Mass Storage class

documents, including Bulk Only, Control/Bulk/Interrupt, Bootability, and UFI Command specifications.

- BOT1.0, SPC-2, SBC-2
- USB 3.0 devices must retain backward compatibility at the Type-A connector to allow Superspeed devices to be used, albeit at a lower speed, with USB 2.0 PCs and allow high-speed devices with their existing cables to be connected to the USB 3.0 Superspeed Type-A connectors.
- USB storage devices must comply with USB 3.0 - Section 11 Interoperability and Power Delivery specs. The following table lists the compatibility matrix for USB 3.0 and USB 2.0. The implication of identifying a host port as supporting USB 3.0 is that both hardware and software support for USB 3.0 is in place; otherwise, the port shall only be identified as a USB 2.0 port.

USB Host Port	USB Device Capability	Connected Mode
USB 2.0	USB 2.0	USB 2.0 high-speed, full-speed, or low-speed
	USB 3.0	USB 2.0 high-speed
USB 3.0	USB 2.0	USB 2.0 high-speed, full-speed, or low-speed
	USB 3.0	USB 3.0 SuperSpeed

- USB storage devices must comply with certification requirement for USB devices and USB storage devices

Note:

Please refer to USB3.0 spec section 3.1.4 USB 3.0 Architecture summary

- USB3.0 Super-speed - 5 Gb/s

- USB2.0 high-speed - 480 Mb/s
- Full-speed - 12 Mb/s
- Low-speed - 1.5 Mb/s

Send comments about this topic to Microsoft

Device.Storage.Hd.Usb3

In this topic:

Device.Storage.Hd.Usb3.Compliance

USB 3.0 storage devices

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB 3.0 storage devices must support industry specifications as indicated below:

- All USB 3.0 Storage devices must be compliant with the USB 3.0 Version 1.0 specification
- Provide unique product identification through each storage end point (BOT, UASP)
  - USB VID/PID

Data Devices must perform as indicated:

- Minimum sequential write speed: 60MB/s
- Minimum sequential read speed: 90MB/s

Send comments about this topic to Microsoft

Device.Storage.Hd.WindowsToGoCapableUSBDrive

Windows To Go capable USB drive feature

In this topic:

Device.Storage.Hd.WindowsToGoCapableUSBDrive.WindowsToGoCapableUSBDrive

Windows To Go capable USB drive

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB boot devices must be USB 3.0 and meet these industry specifications:

- USB 3.0 Version 1.0 specification
- USB BOT specification
- SCSI Block Commands 3 (SBC-3) Specification

USB boot devices must:

- Boot Windows
  - Operate at SuperSpeed when connected to a USB 3.0 port
  - Successfully enter and resume from Sleep (S3) and Hibernate (S4)
- Include in the MS OS Descriptor extended property the value "WindowsBootCapable" DWORD value 1
- Be at least 32 GB in size (20 GB usable)
- Provide unique, consistent product identification
  - USB VID/PID
  - Inquiry Serial Number
  - Inquiry Model Number
- Report FIXED (RMB=0)

- Implement the Block Device Characteristics VPD page
- Not implement IEEE-1667
- Not expose more than one LUN during boot
- Not support the USB Attached SCSI (UAS) protocol
- If the WTG drive exposes multiple functions to the OS:
  - The device must be designed as a composite USB device
  - Composite devices must set DeviceClass, Subclass, and Protocol to 0 in the composite node's device descriptors
  - Must not implement any functions other than WTG Storage and Smartcard
  - The Storage function must be the first function exposed
- Support the following mode pages
  - Mode page code: 0x08 Mode subpage code : 00
- Meet the following performance requirements:
  - Random 4 KB Write IOPs >= 200 (Rotational drives exempt)
  - Random 4 KB Read IOPs >= 2000 (Rotational drives exempt)
  - Read/Write perf should scale linearly in mixed workload random access read/write
  - Sequential write speed >= 80 MB/s
  - Sequential read speed >= 80 MB/s
  - Max I/O Latency < 500 milliseconds
  - Maximum of 16 seconds sum-total of user-perceivable I/O latencies over any 1 hour period of a user-representative workload, where a user-perceivable I/O is defined as having a latency of at least 100 millisecond

Send comments about this topic to Microsoft

Device.Storage.Optical

In this topic:

Device.Storage.Optical.CdRawRecording

Optical drives must support CD RAW recording.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Optical drives must support CD RAW recording mode for CD-R and CD-RW profiles.

Device.Storage.Optical.CommandPerformance

Optical drives must complete Performance Command within allowed time frames.

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Optical drives must complete all commands within the maximum allowed times according to the following table.

COMMAND	Basic certification requirement (msec)	Exception (exception criteria/ msec)
GET CONFIGURATION	20	
GET EVENT STATUS NOTIFICATION	20	
GET PERFORMANCE	20	
INQUIRY	20	
MECHANISM STATUS	20	
MODE SELECT	20	
MODE SENSE	20	
PREVENT ALLOW MEDIUM REMOVAL	20	
READ TOC PMA ATIP	20	
READ BUFFER CAPACITY	20	
READ CAPACITY	20	
READ CD[1,2,4]	500	
READ DISC INFORMATION	50	
READ FORMAT CAPACITIES	20	
READ TRACK INFORMATION	20	
REQUEST SENSE (when not following a command failed with error status)	20	
SEND OPC INFORMATION	60000	For dual layer media, i.e. DVD+R DL, DVD-R DL, BD-RE DL / 70000

SET READ AHEAD	20	
SET STREAMING	20	
START STOP UNIT (Immed=1b)	20	
START STOP UNIT (Eject, Immed=0b)	7000	
START STOP UNIT (Inject, Immed=0b, until media is ready)	25000	DVD-RAM / 40000 For dual layer media, i.e. DVD+R DL, DVD-R DL, BD-RE DL / 30000
SYNCHRONIZE CACHE (Immed=1b)	20	
SYNCHRONIZE CACHE (Immed=0b)	15000	
TEST UNIT READY	20	
BLANK (Immed=1b)	20	
BLANK (Immed=0b)	N/A	
CLOSE TRACK SESSION (Immed=1b)	20	
CLOSE TRACK SESSION (Immed=0b, close logical track or session, do not finalize disc)	65000	DVD+R DL, DVD-R DL / 300000 DVD-R SL, DVD-RW / 180000
CLOSE TRACK SESSION (Immed=0b, finalize disc)	65000	DVD+R DL, 300000 DVD-R SL, DVD-R DL DVD-RW / 900000
FORMAT UNIT (Immediate)	20	
LOAD/UNLOAD MEDIUM	N/A	
READ10[1,2,4]	500	DVD-RAM / 800

READ12 (not streaming)[1,2,4]	500	DVD-RAM / 800
READ12 (streaming)[1,2,4]	100	CD at 1X / 500; CD at 2x / 350; CD at 4x / 200
READ DISC STRUCTURE[6]	20	
READ MEDIA SERIAL NUMBER[2]	500	
REPORT KEY	20	
RESERVE TRACK	N/A	
SEND CUE SHEET	N/A	
SEND DISC STRUCTURE	N/A	
SEND KEY	20	
SET CD SPEED	20	
WRITE 10 (FUA=0) [1,3]	50	
WRITE 12 (FUA=0) [1,3]	50	
WRITE BUFFER	N/A	
ERASE	N/A	
READ BUFFER	N/A	
READ CD MSF[1,2,4]	500	
REPAIR TRACK	N/A	
SEEK10	N/A	
VERIFY 10	N/A	
WRITE AND VERIFY 10	N/A	

"Command completion time" is defined as the time between a command leaving the Microsoft port / miniport driver and the command completion being returned to the Microsoft port / miniport driver. If the command is failed with error status, this time also includes the subsequent Request Sense command leaving the Microsoft port / miniport driver and the Request Sense command completion being returned to the Microsoft port / miniport driver.

All the command execution time performance measurement should be performed on media conforming to media physical layer standard specification from associated committees - i.e. DVD Forum, BDA, DVD+RW Alliance. Also, they should be performed under normal temperature and humidity operational condition as declared in the device specification.

Note: Read-Only drives will be retired from the Windows certification Program on June 01, 2010. Hence, certification requirements and tests will cease to exist for Read-Only drives on June 01, 2010. Partners who wish to receive Windows certification on systems with Read-Only drives would still be able to. However, the Read-Only drive would fall under the "unclassified" category of devices.

[1]: Transfer length for the read and write performance tests is equal or smaller to a single ECC block (32 KB or 64 KB depending of the current media type, i.e. 64 KB for CD and BD, 32 KB for DVD).

[2]: Performance tests may be exercised at any speed reported as supported by the device, including 1x media speed if so reported as supported.

[3]: Drive **must** make use of write buffer and **shall not** delay command completion by any form of media access. If write buffer is full, drive **must** fail the write command with long write in progress sense information (02h/04h/08h).

[4]: The first hundred read I/O commands after media arrival or resume from StandBy power state or Set Cd Speed or Set Streaming are permitted a delay up to a cumulative total of 60 000 msec to complete to allow for additional spin-up time. These commands individually may take any duration up to a limit of 7 000 msec, but the cumulative time to complete all hundred commands shall not exceed 60 000 msec. Only the time between when a read command is sent to the device and that read command is completed by the device is accounted for, the time between two successive read commands is not accounted for (i.e. host delays are not measured).

[6]: The list of disc structure codes is limited to; physical format information (Format = 0x00, Address = 0), DVD-RAM medium status (Format = 0x09, Address = 0), DVD+RW write inhibit DCB (Format = 0x30 Address = 0x57444300), write protection status (Format = 0xC0 Address = 0)7:Dual Layer Write profile will be required on 1 June 2010 for Blu-Ray drives of 9.5 mm height and smaller as well as DVD drives 7 mm height and smaller. This ends the previous exception for these form factors.

Device.Storage.Optical.DriveDefinition

How optical drives are defined for certification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

To be an Optical Drive, the device must be defined as CD (Compact Disc) device, DVD (Digital Versatile Disc or Digital Video Disc) device, BD (Blu-Ray Disc) device or any device which identifies itself as Peripheral Device Type 5 per INCITS's T10's command set SCSI Primary Commands, SPC (any revision).

Device.Storage.Optical.Features

Required optical drive features

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Optical drives must support the required features listed below:

- Core Feature
- Profile List Feature
- Mandatory features per profile
- Removable medium feature from Mt. Fuji 7
- Reporting correct tray status
- Power management feature
- Morphing Feature
- Drive Serial Number Feature
- DVD CSS Feature (0106h)

Device.Storage.Optical.MmcVersion

Optical drives must comply with MMC.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Optical drives must conform to INCITS's T10's command set and MultiMedia Command Set - 6 (MMC-6) when published. Because the publication of MMC-6 has been delayed, Optical Drives must in the interim conform to the combination of INCITS's T10's command set MultiMedia Command Set - 5 (MMC-5) and SFF's Mt. Fuji Commands for Multimedia Devices Version 7 (INF-8090i v7) until the publication of MMC-6. If and when MMC-5 and INF-8090i v7 contradict each other, and the following requirements do not specify explicitly the required behavior, compliance to MMC-5 is required (with the exception of features newly defined in INF-8090i v7).

Device.Storage.Optical.Profiles

Required optical drive profiles

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Optical drives must support the required profiles as listed below:

- CD-ROM
- DVD-ROM
- Removable Disk
- CD-R
- CD-RW
- DVD-R Sequential Recording
- DVD-RW Restricted Overwrite
- DVD-R Dual Layer Sequential Recording
- DVD+RW
- DVD+R
- DVD+R DL

Device.Storage.Optical.RealTimeStreaming

Optical drives must support Real Time Streaming.

Applies to	Windows 10 x64 Windows 10 x86

	Windows Server 2016 Technical Preview x64

Description

Optical Drives must support Real Time Streaming as required according to Profile requirements. For all recordable and rewritable profiles, the following fields shall be set accordingly: Stream Writing (SW)=1b and Write Speed Performance Descriptor (WSPD)=1b.

Send comments about this topic to Microsoft

Device.Storage.Optical.BluRayReader

In this topic:

- Device.Storage.Optical.BluRayReader.Profiles

Device.Storage.Optical.BluRayReader.Profiles

Required profiles for Blu-Ray readers

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Blu-Ray reader drives must support BD-ROM profile.

Send comments about this topic to Microsoft

Device.Storage.Optical.BluRayWriter

In this topic:

- Device.Storage.Optical.BluRayWriter.Profiles

Device.Storage.Optical.BluRayWriter.Profiles

Required profiles for Blu-Ray writers

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Blu-Ray drives that can write must support BD-ROM, BD-R Sequential Recording and BD-RE profiles.

Send comments about this topic to Microsoft

Device.Storage.Optical.Sata

In this topic:

Device.Storage.Optical.Sata.AsynchronousNotification

Asynchronous Notification is required for all SATA connected drives.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Optical drives that connect via the SATA bus must support Asynchronous Notification.

Send comments about this topic to Microsoft

Device.Streaming.Camera.Base

In this topic:

- Device.Streaming.Camera.Base.DirectShow
- Device.Streaming.Camera.Base.KSCategoryVideoCameraRegistration
- Device.Streaming.Camera.Base.MultipleClientAppSupport
- Device.Streaming.Camera.Base.SurpriseRemoval
- Device.Streaming.Camera.Base.UsageIndicator

Device.Streaming.Camera.Base.AVStreamClassInterfaceAndWDM

Camera implementation must be based on AVStream class and WDM, and must meet interface requirements

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64

Description

Camera implemetations must be based on the AVStream class interface and Windows Driver Model (WDM) as defined in the Windows Driver Kit. AVStream is the replacement technology for Stream class interface, which is no longer supported.

Camera drivers must support all of the required pins, properties, and settings as defined in the Windows Driver Kit.

Camera Profiles (If Implemented):

The driver may advertise profiles which are combinations of camera capabilities that are guaranteed to work simultaneously. More information on Camera Profiles can be found in the WDK.

Independent Pins:

All pins must be able to operate independently and in combination according to the capabilities listed in the Profiles advertised by the device. If the device does not support Profiles, then concurrent streaming for all advertised capabilities for all media types must be supported unless specifically called out as not supported by setting the Photo and PhotoSequence exclusivity flags in the driver INF. See additional documentation about these flags here:

- https://msdn.microsoft.com/en-us/library/windows/hardware/jj553707(v=vs.85).aspx

USB based cameras must be USB Video Class (UVC) compliant as defined in **Device.Streaming.Camera.UVC.UVCDriver** and **Device.Streaming. Camera.UVC.USBClassDriver**.

ErrorConditions

Error Conditions include (but are not limited to) forced invalid pin connections, invalid property sets, buffers with invalid data, null pointers, and error conditions from drivers above or below on the stack

Device.Streaming.Camera.Base.PnPandPowerPolicies

Camera implementation must comply with Microsoft PNP and power policies.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

Cameras must comply with Microsoft PNP and power policies and also support connected stand by configurations.

More info can be found: http://msdn.microsoft.com/en-us/library/Windows/Hardware/dn584781(v=vs.85).aspx

Device.Streaming.Camera.Base.MediaFoundation

Camera implementation must support Media Foundation architecture.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

Cameras must work with Microsoft Media Foundation in order to support legacy and Windows Store applications.

Device.Streaming.Camera.Base.DirectShow

Camera implementation must support DirectShow.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

Cameras must work with Microsoft DirectShow in order to support legacy applications.

Device.Streaming.Camera.Base.KSCategoryVideoCameraRegistration

Cameras must register under KSCategory_Video_Camera

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

Cameras must register under KSCategory_Video_Camera for Windows to detect camera

Device.Streaming.Camera.Base.MultipleClientAppSupport

Camera implementation must support multiple KS filter instances and independent streaming

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

Multiple filter instance of the same device:

Each camera must allow multiple filter instances to be created. Devices may only allow one filter instance to actively stream.

Device.Streaming.Camera.Base.SurpriseRemoval

Hot-pluggable cameras must support surprise removal.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

All hot-pluggable cameras must support their surprise removal from the host bus.

Device.Streaming.Camera.Base.UsageIndicator

Cameras must have a visual indicator to indicate usage

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

A camera must have a visual indicator that indicates when it is streaming video. The visual indicator must be on when device is capturing video and off when the device is not capturing video.

A hardware usage indicator (physical LED) is strongly recommended. In lieu of a physical LED, the following regkey may be set:

Path: HKLM\SOFTWARE\Microsoft\OEM\Device\Capture

Entry: NoPhysicalCameraLED (REG_DWORD)

0x1: Turn on the feature (= No Physical camera LED on the system)

0x0: Default. Turn off the feature (= Physical camera LED on the system)

When this is set, the operating system will show UI indicating when the camera is streaming.

Send comments about this topic to Microsoft

Device.Streaming.Camera.UVC

In this topic:

Device.Streaming.Camera.UVC.USBClassDriver

USB Streaming video cameras must comply with USB Video Class specifications and work with the Microsoft UVC driver (USBVideo.sys)

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64

Description

All USB streaming video Cameras must be compatible with Microsoft USB Video class driver (USBVideo.sys).

Device.Streaming.Camera.UVC.UVCDriver

USB streaming video cameras must comply with USB Video Class specifications

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64

Description

All USB streaming video cameras, including those not natively using the Microsoft USB Video Class driver (USBVideo.sys), must comply with the USB Video Class specifications. At a minimum, all mandatory properties and commands must be implemented. All implemented commands must comply with the specifications.

Non Microsoft Driver (If Implemented)

USB Camera implementations that do not use the Microsoft USB Video Class driver must still be compatible with USBVideo.sys (per **Device.Streaming.Camera.UVC.USBClassDriver**)

Native H264 Support (If Implemented)

If H.264 format is supported natively by the camera, the implementation must be compliant with the USB Video Class specification. For reference, see: http://go.microsoft.com/fwlink/?LinkId=233063

At minimum 2 pins must be supported:

1. A preview pin on the same device with uncompressed output type YUY2 and/or NV12
2. H.264 format must be on a separate video capture pin

Send comments about this topic to Microsoft

Device.Streaming.HMFT

In this topic:

Device.Streaming.HMFT.Decoding

Hardware Media Foundation Transform (HMFT) must support video decoding.

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

Supported Formats

HMFT video decoder is only supported for MPEG-4 Part 2 and MJPEG. Both of these formats are if-implemented.

Media Foundation Compliance

The video decoder Hardware Media Foundation Transform (HMFT) must fully comply with the following Media Foundation Transform (MFT) interfaces:

- IMFTransform
- IMFMediaEventGenerator
- IMFShutdown
- IMFQualityAdvise
- IMFRealTimeClientEx

The HMFT video decoder must use Media Foundation work queue support (no thread creation) via **IMFRealTimeCLientEx::SetWorkQueue**. This ensures the following:

- Multimedia Class Scheduler Service (MMCSS) support for playback and capture/encode
- Improved core scalability

The HMFT video decoder must support **IMF2DBuffer2** for enhanced security, but also work with input buffers of type **IMF2DBuffer**, and **IMFMediaBuffer** in that order of preference.

DirectX Rendering

The video decoder HMFT must support both DirectX (DX) 9 and, DX11 devices, and it must avoid copies in or out of DX11 components.

On MFT_Set_D3D_Manager, the video decoder HMFT must first query for DirectX Graphics Infrastructure (DXGI) Manager and then query for D3D9 Manager if DXGI Manager is not found.

The HMFT video decoder must support system memory output because some transforms in the pipeline may support only system memory buffers.

If the HMFT video decoder is based on a GPU, it must support DX11.

Memory Usage

The HMFT video decoder must be an asynchronous MFT. This reduces the memory working set by about 50 percent.

Trusted Merit Certification and Verification

The video decoder HMFT must support the Trusted Merit Certification and Verification process, as defined across the Windows Hardware Certification Kit.

Each HMFT video decoder must be provided as a separate binary and must be individually signed.

HMFT video decoders must not advertise support for more than one compression standard.

All HMFTs must set the following Media Foundation attributes while registering the MFT with the system:

- MFT_ENUM_HARDWARE_URL_Attribute
- MFT_ENUM_HARDWARE_VENDOR_ID_Attribute

Format Requirements

HMFT video decoders must not advertise support for inbox formats supported by DirectX Video Acceleration (DXVA) (H.264, H.264, WMV, MPEG2).

If implemented: the HMFT video decoder for MPEG-4 Part 2 must support Simple and Advanced Simple Profile (If Global Motion Compensation (GMC) is not supported, then the media type must be rejected to allow the software decoder to be used for playback), and all levels.

- The decoder must be fully conformant to specifications that are defined for the format.
- In addition, the MPEG-4 Part 2 decoder must fully support H.263 baseline content and advertise support for this media type.

In addition to the preceding requirements, we recommend that the decoder support post-processing for deblocking and deringing.

Vendors may provide other HMFTs video decoders for formats that are not supported inbox, but there are no verification tests or logo certification available.

Note: The recommendations and requirements that are defined in this document apply to all formats.

Functionality

The video decoder HMFT must support the following functionalities:

- Dynamic format and resolution changes
- Trick modes (playback rate control, thinning mode) and seek

Interlace Support

The HMFT video decoder must support the input format for both interlaced and progressive bit streams. It must not de-interlace. It may support inverse telecine (IVTC).

Multiple Instances

The HMFT video decoder must support multiple instances of the decoder in parallel (both in-process and out of process) to enable multiple concurrent video playback streams in the same or different applications.

Design Notes

The HMFT video decoder must be installed and uninstalled through a device driver that meets Windows security requirements. The driver must not cause the operating system to crash or hang, and must disallow memory violations.

Each HMFT component must be a separate binary, individually certified and signed.

Device.Streaming.HMFT.Encoding

Hardware Media Foundation Transform (HMFT) must support video encoding.

Applies to	Windows 10 for desktop editions x86 Windows 10 for desktop editions x64 Windows Server 2016 Technical Preview x64

Description

A. H.264 Encode

If your hardware supports H.264 Encode, you must:

A.1 Input Format Support

The encoder must support NV12 format for input frames. Optionally, the encoder can support IYUV and YUY2 formats.

The encoder must implement support for the IMFMediaBuffer, IMF2DBuffer, and IMF2DBuffer2 media buffer interfaces.

A.2 Profile Support

The encoder must be capable of creating output content that is conformant to the following profiles:

Baseline profile

Constrained Baseline profile

Main profile

Constrained High profile

The encoder must be capable of generating the appropriate header syntax elements to signal that it is encoded to the above listed profiles.

A.3 Interface Support Requirements

The H.264 encoder must support the following interfaces:

A.3.1 IMFTransform

Implementation of the following methods of the IMFTransform interface is required:

A.3.1.1 IMFTransform::ProcessEvent::GetAttributes

A.3.1.2 IMFTransform::GetInputAvailableType

A.3.1.3 IMFTransform::GetInputCurrentType

A.3.1.4 IMFTransform::GetInputStatus

A.3.1.5 IMFTransform::GetInputStreamInfo

A.3.1.6 IMFTransform::GetOutputAvailableType

A.3.1.7 IMFTransform::GetOutputCurrentType

A.3.1.8 IMFTransform::GetOutputStatus

A.3.1.9 IMFTransform::GetOutputStreamInfo

A.3.1.10 IMFTransform::GetStreamCount

A.3.1.11 IMFTransform::GetStreamLimits

A.3.1.12 IMFTransform::ProcessEvent

Must handle the MEEncodingParameters event

A.3.1.13 IMFTransform::ProcessMessage

A.3.1.14 IMFTransform::ProcessInput

A.3.1.15 IMFTransform::ProcessOutput

A.3.1.16 IMFTransform::SetInputType

A.3.1.17 IMFTransform::SetOutputType

A.3.1.18 IMFTransform::SetOutputBounds

A.3.2 ICodecAPI

The encoder must support certain methods and property GUIDs of the ICodecAPI interface.

A.3.2.1 Required Methods

Implementation of the following methods of the ICodecAPI interface is required:

A.3.2.1.1 IsSupported

A.3.2.1.2 GetValue

A.3.2.1.3 SetValue

A.3.2.1.4 GetParameterRange

A.3.2.1.5 GetParameterValues

A.3.2.2 Required Properties

Support for the following ICodecAPI property GUIDs is required

A.3.2.2.1 CODECAPI_AVEncCommonRateControlMode

A.3.2.2.2 CODECAPI_AVEncCommonQuality

A.3.2.2.3 CODECAPI_AVEncCommonMeanBitRate

A.3.2.2.4 CODECAPI_AVEncCommonMaxBitRate

A.3.2.2.5 CODECAPI_AVEncCommonBufferSize

A.3.2.2.6 CODECAPI_AVEncMPVGOPSize

A.3.2.2.8 CODECAPI_AVLowLatencyMode

A.3.2.2.9 CODECAPI_AVEncCommonQualityVsSpeed

A.3.2.2.10 CODECAPI_AVEncVideoForceKeyFrame

A.3.2.2.11 CODECAPI_AVEncVideoEncodeQP

A.3.2.2.12 CODECAPI_AVEncVideoTemporalLayerCount

A.3.2.2.13 CODECAPI_AVEncVideoSelectLayer

A.3.2.3 Optional Properties

Support for the following ICodecAPI property GUIDs is optional.

A.3.2.3.1 CODECAPI_AVEncH264CABACEnable

A.3.2.3.2 CODECAPI_AVEncMPVDefaultBPictureCount

A.3.2.3.3 CODECAPI_AVEncVideoEncodeFrameTypeQP

A.3.2.3.4 CODECAPI_AVEncSliceControlMode

A.3.2.3.5 CODECAPI_AVEncSliceControlSize

A.3.2.3.6 CODECAPI_AVEncVideoMaxNumRefFrame

A.3.2.3.7 CODECAPI_AVEncVideoMeanAbsoluteDifference

A.3.2.3.8 CODECAPI_AVEncVideoROIEnabled

A.3.2.3.9 CODECAPI_AVEncVideoMinQP

A.3.2.3.10 CODECAPI_AVEncVideoMaxQP

A.3.2.3.11 CODECAPI_AVEncVideoLTRBufferControl

A.3.2.3.12 CODECAPI_AVEncVideoMarkLTRFrame

A.3.2.3.13 CODECAPI_AVEncVideoUseLTRFrame

A.3.2.3.14 CODECAPI_AVEncLowPowerEncoder

A.3.2.3.15 CODECAPI_AVScenarioInfo

A.3.2.3.16 CODECAPI_AVEncMPVGOPSizeMin

A.3.2.3.17 CODECAPI_AVEncMPVGOPSizeMax

A.3.2.3.18 CODECAPI_AVEncVideoMaxTem

A.3.2.3.19 CODECAPI_AVEncMaxFrameRate

A.3.2.3.20 CODECAPI_VideoEncoderDisplayContentType

A.3.2.3.22 CODECAPI_AVEncEnableVideoProcessing

A.3.2.3.23 CODECAPI_AVEncVideoGradualIntraRefresh

A.3.2.4 Optional Property Notes

If your H.264 encoder implements Long Term Reference frames, then the following CodecAPI interfaces should be implemented (completed within one frame delay):

CODECAPI_AVEncVideoLTRBufferControl

CODECAPI_AVEncVideoMarkLTRFrame

CODECAPI_AVEncVideoUseLTRFrame

If your H.264 encoder implements Long Term Reference frames, then the IMFSample attribute LongTermReferenceFrameInfo must be supported.

A.4 MF Media Type Attribute Requirements

Support for the following MF media type attributes is required:

A.4.1 MF_MT_INTERLACE_MODE

A.4.2 MF_MT_MINIMUM_DISPLAY_APERTURE

A.4.3 MF_MT_SUBTYPE

A.4.4 MF_MT_MINIMUM_DISPLAY_APERTURE

A.4.5 MF_MT_FRAME_RATE

A.4.6 MF_MT_FRAME_SIZE

A.4.7 MF_MT_AVG_BITRATE

A.4.8 MF_MT_MPEG2_PROFILE (MF_MT_VIDEO_PROFILE)

A.4.9 MF_MT_MPEG2_LEVEL (MF_MT_VIDEO_LEVEL)

A.4.10 MF_MT_PIXEL_ASPECT_RATIO

A.4.11 MF_MT_VIDEO_NOMINAL_RANGE

A.4.12 MF_MT_VIDEO_PRIMARIES

A.4.13 MF_MT_TRANSFER_FUNCTION

A.4.14 MF_MT_YUV_MATRIX

A.4.15 MF_MT_VIDEO_CHROMA_SITING

A.4.16 MF_NALU_LENGTH_SET

A.5 MF Sample Attribute Support Requirements

A.5.1 Required Sample Attributes

Subject to whether an ICodecAPI or MF media type attribute is set, the following sample attributes must be present on input or output samples:

A.5.1.1 MFSampleExtension_MeanAbsoluteDifference

The MFSampleExtension_MeanAbsoluteDifference sample attribute is required on output samples if the CODECAPI_AVEncVideoMeanAbsoluteDifference property has been set to a non-zero value. Note that because CODECAPI_AVEncVideoMeanAbsoluteDifference is an optional property, support for this attribute is only required for the case where CODECAPI_AVEncVideoMeanAbsoluteDifference is supported.

A.5.1.2 MFSampleExtension_LongTermReferenceFrameInfo

The MFSampleExtension_LongTermReferenceFrameInfo sample attribute is required on output samples if the CODECAPI_AVEncVideoLTRBufferControl has indicated that one or more long-term reference frames could be present. Note that because CODECAPI_AVEncVideoLTRBufferControl is an optional property, support for this attribute is only required for the case where CODECAPI_AVEncVideoLTRBufferControl is supported.

A.5.1.3 MF_NALU_LENGTH_INFORMATION

The MF_NALU_LENGTH_INFORMATION sample attribute is required on output samples if the MF_NALU_LENGTH_SET media type attribute has been set to a non-zero value.

A.5.1.4 MFSampleExtension_ROIRectangle

The input sample may contain an MFSampleExtension_ROIRectangle sample attribute if the CODECAPI_AVEncVideoROIEnabled property has been set to a non-zero value. Note that because CODECAPI_AVEncVideoROIEnabled is an optional property, support for this attribute is only required for the case where CODECAPI_AVEncVideoROIEnabled is supported.

A.5.2 Optional Sample Attributes

The following input sample attributes may be present. It is optional whether the encoder utilizes the information contained by these attributes.

A.5.2.1 MFSampleExtension_DirtyRects

A.5.2.2 MFSampleExtension_MoveRegions

A.6 MFT Attributes

The encoder is required to support the IMFTransform::GetAttributes which will return the IMFAttributes interface which allows access to MFT attributes. Support for handling the following MFT MF attributes is required:

A.6.1 MFT_ENUM_HARDWARE_URL_Attribute

A.6.2 MFT_ENUM_HARDWARE_VENDOR_ID_Attribute

A.6.3 MFT_ENCODER_SUPPORTS_CONFIG_EVENT

A.6.4 MF_VIDEO_MAX_MB_PER_SEC

A.6.5 MFT_GFX_DRIVER_VERSION_ID_Attribute

A.6.6 MFT_ENCODER_ERROR

A.7 Functionality Support Requirements

The H.264 encoder must support the following functionality:

A.7.1 Peak-constrained VBR rate control mode

The encoder will use the values of the CODECAPI_AVEncCommonMeanBitRate, CODECAPI_AVEncCommonMaxBitRate, and CODECAPI_AVEncCommonBufferSize properties to ensure that the theoretical buffer is not violated at the peak bitrate (CODECAPI_AVEncCommonMaxBitRate) while converging to a bitrate of CODECAPI_AVEncCommonMeanBitRate for the entire video clip.

A.7.2 Quality-based VBR rate control mode

The encoder will use the value of CODECAPI_AVEncCommonQuality or CODECAPI_AVEncVideoEncodeQP to achieve constant quality of the compressed video. The encoder will use a constant QP for all frames.

A.7.3 CBR rate control mode

The encoder will use the values of the CODECAPI_AVEncCommonMeanBitRate, or MF_MT_AVG_BITRATE and CODECAPI_AVEncCommonBufferSize properties to ensure that the theoretical buffer is not violated at the specified bitrate (CODECAPI_AVEncCommonMeanBitRate or MF_MT_AVG_BITRATE).

A.7.4 GOP distance

The encoder will produce a key-frame at the frame interval specified by the value of the CODECAPI_AVEncMPVGOPSize property. A value of IntMAX for this property indicates that only the first frame should be a key frame.

A.7.5 Frame-based QP control

Must support at minimum QP range from 20 to 40. Must return an error when an invalid value (e.g. 52) is set.

A.7.6 Forced Key-frame control

IDR must be preceded with SPS/PPS. When the number of temporal layers > 1, key frame must be inserted in the next base temporal layer frame in order to preserve the temporal structure. For a single layer bitstream, key frame must be inserted in the next frame.

A.7.7 Quality versus Speed control

A.7.8 Low-latency control

The encoder must support the ability to be set to low latency mode. In this mode, the encoder should not buffer any input samples. It must be capable of producing an output sample for each input sample.

The encoder also must either force POC type 2 or set the VUI bitstream_restriction flag to TRUE and VUI num_reorder_frames to 0.

A.7.8 Temporal Layer Support

It is required that the encoder support hierarchical P frames encoding and be capable of generating bitreams of 1, 2, or 3 temporally-independent layers.

A.7.9 Long Term Reference Frame Support

If your H.264 encoder implements Long Term Reference frames, then the following CodecAPI interfaces should be implemented (completed within one frame delay):

CODECAPI_AVEncVideoLTRBufferControl

CODECAPI_AVEncVideoMarkLTRFrame

CODECAPI_AVEncVideoUseLTRFrame

A.8 Asynchronous MFT Support

It is recommended that you implement an asynchronous MFT. An asynchronous MFT requires that you implement the IMFMediaEventGenerator and IMFShutdown interfaces.

A.9 IMFRealTimeClientEx

Registration with MMCS via IMFRealTimeCLientEx::SetWorkQueue is critical to meet performance goals, therefore, the implementation of the IMFRealTimeClientEx interface is recommended.

A.9 Multiple Instance Requirements

It is required that your H.264 encoder must support a minimum of 4 concurrent instances. The encoder should not put any limitation on the number of instances. It should be bounded by the memory usage.

These instances may be in the same process or in different processes.

A.10 Merit Validation

It is required that your H.264 encoder supports the trusted merit verification process.

It is required that your H.264 encoder be a separate binary, individually certified and signed.

A.11 Additional Requirements

Your H.264 encoder must work with the Windows MP4 file sink.

Your H.264 encoder must implement the proper order of encoding configuration.

Your H.264 encoder must be capable of producing valid bitstreams with up to 3 temporal layers.

Your H.264 encoder prefix NALU prepends each slice and is syntax correct with bitstreams with 1-3 temporal layers.

Your H.264 encoder must send one batched GPU request per frame.

Your H.264 encoder must insert an AUD NALU before each frame.

Your H.264 encoder must work correctly in session 0.

A.12 Installation

It is required that your H.264 encoder is registered and unregistered along with the device driver used in the encoder.

A.113 Dynamic Format Change

It is required that your H.264 encoder supports dynamic format changes of the following within:

Profile Change

Resolution and Frame Rate Change

Add and Delete Temporal Layers

Maximum allowed number of reference frames

When multiple changes are requested before an output frame is produced, the last change should be honored.

B. VC-1 Encode

If your hardware supports VC-1 Encode, you must:

B.1 On input:

B.1.1 Support NV12

B.1.2 If your hardware supports it:

B.1.2.1 Support IYUV and YUY2

B.1.3 Support input buffers of (and query in this order):

B.1.3.1 IMF2DBuffer2

B.1.3.2 IMF2DBuffer

B.1.3.3 IMFMediaBuffer

B.2 On output:

B.2.1 Support Simple profile

B.2.2 Support Main profile

B.2.3 Support Advanced profile

B.3 Your VC-1 encoder must expose the following interfaces:

B.3.1 IMFTransform

B.3.2 ICodecAPI

B.3.2.1 ICodecAPI requires the following functions to be implemented:

B.3.2.1.1 IsSupported

B.3.2.1.2 GetValue

B.3.2.1.3 SetValue

B.3.2.2 ICodecAPI requires the following properties to be supported:

B.3.2.2.1 Peak-constrained VBR mode

B.3.2.2.2 Quality-based VBR mode

B.3.2.2.3 CBR encoding

B.3.2.2.4 GOP distance

B.3.2.2.5 Frame-QP control

B.3.2.2.6 Adaptive bitrate control

B.3.2.2.7 Force-Key-frame control

B.3.2.2.8 QualityVsSpeed control

B.3.2.2.9 Low-latency encoding

B.3.2.3 It is recommended that you additionally implement the following functions:

B.3.2.3.1 GetDefaultValue

B.3.2.3.2 GetParameterRange

B.3.2.3.3 GetParameterValues

B.3.2.3.4 IsModifiable

B.3.2.3.5 SetAllDefaults

B.3.3 IMFAttributes

B.3.3.1 Via IMFTransform::GetAttributes

B.3.3.2 The two required attributes are:

B.3.3.2.1 MFT_ENUM_HARDWARE_URL_Attribute

B.3.3.2.2 MFT_ENUM_HARDWARE_VENDOR_ID_Attribute

B.3.4 It is recommended that you implement an asynchronous MFT.

B.3.4.1 Asynchronous MFTs require the following additional interfaces:

B.3.4.1.1 IMFMediaEventGenerator

B.3.4.1.2 IMFShutdown

B.3.4.2 Asynchronous MFTs are encouraged to avoid creating threads and are recommended to use the MF Thread Pool.

B.3.4.2.1 Registration with MMCS via IMFRealTimeCLientEx::SetWorkQueue is critical to meet performance goals.

B.3.5 IMFRealTimeClientEx is recommended

B.4 Encoding Settings

B.4.1 Through input media type negotiation, the VC-1 Encoder must support:

B.4.1.1 MF_MT_INTERLACE_MODE

B.4.1.1.1 The encoder must preserve interlace from input to output or reject interlace.

B.4.1.2 MF_MT_MINIMUM_DISPLAY_APERTURE

B.4.2 Through output media type negotiation, the VC-1 Encoder must support:

B.4.2.1 MF_MT_SUBTYPE

B.4.2.2 MF_MT_FRAME_SIZE

BC.4.2.3 MF_MT_FRAME_RATE

B.4.2.4 MF_MT_AVG_BITRATE

B.4.2.5 MF_MT_PIXEL_ASPECT_RATIO

B.4.3 It is recommended that your VC-1 Encoder supports:

B.4.3.1. B frame encoding

B.5 Multiple Instances

B.5.1 It is required that your VC-1 encoder must support a minimum of 3 concurrent instances. Encoder should not put any limitation on number of instances. It should be bounded by the memory usage.

B.5.1.1 These instances may be in the same process or in different processes.

B.6 Merit Validation

B.6.1 It is required that your VC-1 encoder supports the trusted merit verification process.

B.6.2 It is required that your VC-1 encoder be a separate binary, individually certified and signed.

B.7 Additional Requirements

B.7.1 Your VC-1 encoder must work with the Windows ASF file sink.

B.7.2 Your VC-1 encoder must implement proper order of encoding configuration.

B.7.3 Your VC-1 encoder must work correctly in session 0.

B.8 Installation

B.8.1 It is required that your VC-1 encoder is registered and unregistered along with the device driver used in the encoder.

C. H.265 Encode

If your hardware supports H.265 Encode, you must:

C.1 Input Format Support

The encoder must support NV12 format for input frames. Optionally, the encoder can support IYUV and YUY2 formats.

The encoder must implement support for the IMFMediaBuffer, IMF2DBuffer, and IMF2DBuffer2 media buffer interfaces.

C.2 Profile Support

The encoder must be capable of creating output content that is conformant to the Main profile.

The encoder must be capable of generating the appropriate header syntax elements to signal that it is encoded to the Main profile.

C.3 Interface Support Requirements

The H.265 encoder must support the following interfaces:

C.3.1 IMFTransform

Implementation of the following methods of the IMFTransform interface is required:

C.3.1.1 IMFTransform::ProcessEvent::GetAttributes

C.3.1.2 IMFTransform::GetInputAvailableType

C.3.1.3 IMFTransform::GetInputCurrentType

C.3.1.4 IMFTransform::GetInputStatus

C.3.1.5 IMFTransform::GetInputStreamInfo

C.3.1.6 IMFTransform::GetOutputAvailableType

C.3.1.7 IMFTransform::GetOutputCurrentType

C.3.1.8 IMFTransform::GetOutputStatus

C.3.1.9 IMFTransform::GetOutputStreamInfo

C.3.1.10 IMFTransform::GetStreamCount

C.3.1.11 IMFTransform::GetStreamLimits

C.3.1.12 IMFTransform::ProcessEvent

Must handle the MEEncodingParameters event:

C.3.1.13 IMFTransform::ProcessMessage

C.3.1.14 IMFTransform::ProcessInput

C.3.1.15 IMFTransform::ProcessOutput

C.3.1.16 IMFTransform::SetInputType

C.3.1.17 IMFTransform::SetOutputType

C.3.1.18 IMFTransform::SetOutputBounds

C.3.2 ICodecAPI

The encoder must support certain methods and property GUIDs of the ICodecAPI interface.

A.3.2.1 Required Methods

Implementation of the following methods of the ICodecAPI interface is required:

C.3.2.1.1 IsSupported

C.3.2.1.2 GetValue

C.3.2.1.3 SetValue

C.3.2.1.4 GetParameterRange

C.3.2.1.5 GetParameterValues

C.3.2.2 Required Properties

Support for the following ICodecAPI property GUIDs is required:

C.3.2.2.1 CODECAPI_AVEncCommonRateControlMode

C.3.2.2.2 CODECAPI_AVEncCommonQuality

C.3.2.2.3 CODECAPI_AVEncCommonMeanBitRate

C.3.2.2.4 CODECAPI_AVEncCommonMaxBitRate

C.3.2.2.5 CODECAPI_AVEncCommonBufferSize

C.3.2.2.6 CODECAPI_AVEncMPVGOPSize

C.3.2.2.8 CODECAPI_AVLowLatencyMode

C.3.2.2.10 CODECAPI_AVEncVideoForceKeyFrame

C.3.2.2.11 CODECAPI_AVEncVideoEncodeQP

C.3.2.3 Optional Properties

Support for the following ICodecAPI property GUIDs is optional.

C.3.2.3.1 CODECAPI_AVEncCommonQualityVsSpeed

C.3.2.3.2 CODECAPI_AVEncVideoTemporalLayerCount

C.3.2.3.3 CODECAPI_AVEncVideoSelectLayer

C.3.2.3.4 CODECAPI_AVEncMPVDefaultBPictureCount

C.3.2.3.5 CODECAPI_AVEncVideoEncodeFrameTypeQP

C.3.2.3.6 CODECAPI_AVEncSliceControlMode

C.3.2.3.7 CODECAPI_AVEncSliceControlSize

C.3.2.3.8 CODECAPI_AVEncVideoMaxNumRefFrame

C.3.2.3.9 CODECAPI_AVEncVideoMeanAbsoluteDifference

C.3.2.3.10 CODECAPI_AVEncVideoROIEnabled

C.3.2.3.11 CODECAPI_AVEncVideoMinQP

C.3.2.3.12 CODECAPI_AVEncVideoMaxQP

C.3.2.3.13 CODECAPI_AVEncVideoLTRBufferControl

C.3.2.3.14 CODECAPI_AVEncVideoMarkLTRFrame

C.3.2.3.15 CODECAPI_AVEncVideoUseLTRFrame

C.3.2.3.16 CODECAPI_AVEncLowPowerEncoder

C.3.2.3.17 CODECAPI_AVScenarioInfo

C.3.2.3.18 CODECAPI_AVEncMPVGOPSizeMin

C.3.2.3.19 CODECAPI_AVEncMPVGOPSizeMax

C.3.2.3.20 CODECAPI_AVEncVideoMaxTem

C.3.2.3.21 CODECAPI_AVEncMaxFrameRate

C.3.2.3.22 CODECAPI_VideoEncoderDisplayContentType

C.3.2.3.23 CODECAPI_AVEncEnableVideoProcessing

C.3.2.3.24 CODECAPI_AVEncVideoGradualIntraRefresh

C.3.2.4 Optional Property Notes

If your H.265 encoder implements Long Term Reference frames, then the following CodecAPI interfaces should be implemented (completed within one frame delay):

CODECAPI_AVEncVideoLTRBufferControl

CODECAPI_AVEncVideoMarkLTRFrame

CODECAPI_AVEncVideoUseLTRFrame

If your H.265 encoder implements Long Term Reference frames, then the IMFSample attribute LongTermReferenceFrameInfo must be supported.

C.4 MF Media Type Attribute Requirements

Support for the following MF media type attributes is required:

C.4.1 MF_MT_INTERLACE_MODE

C.4.2 MF_MT_MINIMUM_DISPLAY_APERTURE

C.4.3 MF_MT_SUBTYPE

C.4.4 MF_MT_MINIMUM_DISPLAY_APERTURE

C.4.5 MF_MT_FRAME_RATE

C.4.6 MF_MT_FRAME_SIZE

C.4.7 MF_MT_AVG_BITRATE

C.4.8 MF_MT_MPEG2_PROFILE (MF_MT_VIDEO_PROFILE)

C.4.9 MF_MT_MPEG2_LEVEL (MF_MT_VIDEO_LEVEL)

C.4.10 MF_MT_PIXEL_ASPECT_RATIO

C.4.11 MF_MT_VIDEO_NOMINAL_RANGE

C.4.12 MF_MT_VIDEO_PRIMARIES

C.4.13 MF_MT_TRANSFER_FUNCTION

C.4.14 MF_MT_YUV_MATRIX

C.4.15 MF_MT_VIDEO_CHROMA_SITING

C.4.16 MF_NALU_LENGTH_SET

C.5 MF Sample Attribute Support Requirements

C.5.1 Required Sample Attributes

Subject to whether an ICodecAPI or MF media type attribute is set, the following sample attributes must be present on input or output samples:

C.5.1.1 MFSampleExtension_MeanAbsoluteDifference

The MFSampleExtension_MeanAbsoluteDifference sample attribute is required on output samples if the CODECAPI_AVEncVideoMeanAbsoluteDifference property has been set to a non-zero value. Note that because CODECAPI_AVEncVideoMeanAbsoluteDifference is an optional property, support for this attribute is only required for the case where CODECAPI_AVEncVideoMeanAbsoluteDifference is supported.

C5.1.2 MFSampleExtension_LongTermReferenceFrameInfo

The MFSampleExtension_LongTermReferenceFrameInfo sample attribute is required on output samples if the CODECAPI_AVEncVideoLTRBufferControl has indicated that one or more long-term reference frames could be present. Note that because CODECAPI_AVEncVideoLTRBufferControl is an

optional property, support for this attribute is only required for the case where CODECAPI_AVEncVideoLTRBufferControl is supported.

C5.1.3 MF_NALU_LENGTH_INFORMATION

The MF_NALU_LENGTH_INFORMATION sample attribute is required on output samples if the MF_NALU_LENGTH_SET media type attribute has been set to a non-zero value.

C.5.1.4 MFSampleExtension_ROIRectangle

The input sample may contain an MFSampleExtension_ROIRectangle sample attribute if the CODECAPI_AVEncVideoROIEnabled property has been set to a non-zero value. Note that because CODECAPI_AVEncVideoROIEnabled is an optional property, support for this attribute is only required for the case where CODECAPI_AVEncVideoROIEnabled is supported.

C.5.2 Optional Sample Attributes

The following input sample attributes may be present. It is optional whether the encoder utilizes the information contained by these attributes.

C.5.2.1 MFSampleExtension_DirtyRects

The MFSampleExtension_DirtyRects attribute is a BLOB type with the following structure:

typedef struct _DIRTYRECT_INFO

{

UINT FrameNumber;

UINT NumDirtyRects;

RECT DirtyRects[1];

} DIRTYRECT_INFO;

This attribute is intended to be used for compressing screen content and allows the window manager to pass information about the dirty rectangles in the screen frame.

C.5.2.2 MFSampleExtension_MoveRegions

The MFSampleExtension_MoveRegions attribute is a BLOB type with the following structure:

typedef struct _MOVE_RECT

{

POINT SourcePoint;

RECT DestRect;

} MOVE_RECT;

typedef struct _MOVEREGION_INFO

{

UINT FrameNumber;

UINT NumMoveRegions;

MOVE_RECT MoveRegions[1];

} MOVEREGION_INFO;

C.6 MFT Attributes

The encoder is required to support the IMFTransform::GetAttributes which will return the IMFAttributes interface which allows access to MFT attributes. Support for handling the following MFT MF attributes is required:

A.6.1 MFT_ENUM_HARDWARE_URL_Attribute

C.6.2 MFT_ENUM_HARDWARE_VENDOR_ID_Attribute

C.6.3 MFT_ENCODER_SUPPORTS_CONFIG_EVENT

C.6.4 MF_VIDEO_MAX_MB_PER_SEC

C.6.5 MFT_GFX_DRIVER_VERSION_ID_Attribute

C.6.6 MFT_ENCODER_ERROR

C.7 Functionality Support Requirements

The H.265 encoder must support the following functionality:

C.7.1 Peak-constrained VBR ratecontrol mode

The encoder will use the values of the CODECAPI_AVEncCommonMeanBitRate, CODECAPI_AVEncCommonMaxBitRate, and CODECAPI_AVEncCommonBufferSize properties to ensure that the theoretical buffer is not violated at the peak bitrate (CODECAPI_AVEncCommonMaxBitRate) while converging to a bitrate of CODECAPI_AVEncCommonMeanBitRate for the entire video clip.

C.7.2 Quality-based VBR ratecontrol mode

The encoder will use the value of CODECAPI_AVEncCommonQuality or CODECAPI_AVEncVideoEncodeQP to achieve constant quality of the compressed video. The encoder will use a constant QP for all frames.

C.7.3 CBR ratecontrol mode

The encoder will use the values of the CODECAPI_AVEncCommonMeanBitRate, or MF_MT_AVG_BITRATE and CODECAPI_AVEncCommonBufferSize properties to ensure that the theoretical buffer is not violated at the specified bitrate (CODECAPI_AVEncCommonMeanBitRate or MF_MT_AVG_BITRATE).

C.7.4 GOP distance

The encoder will produce a key-frame at the frame interval specified by the value of the CODECAPI_AVEncMPVGOPSize property. A value of IntMAX for this property indicates that only the first frame should be a key frame.

C.7.5 Frame-based QP control

Must support at minimum QP range from 20 to 40. Must return an error when an invalid value (e.g. 52) is set.

C.7.6 Forced Key-frame control

IDR must be preceded with SPS/PPS. When the number of temporal layers > 1, key frame must be inserted in the next base temporal layer frame in order to preserve the temporal structure. For a single layer bitstream, key frame must be inserted in the next frame.

C.7.7 Quality versus Speed control

C.7.8 Low-latency control

The encoder must support the ability to be set to low latency mode. In this mode, the encoder should not buffer any input samples. It must be capable of producing an output sample for each input sample.

The encoder also must either force POC type 2 or set the VUI bitstream_restriction flag to TRUE and VUI num_reorder_frames to 0.

C.7.8 Temporal Layer Support

It is required that the encoder support hierarchical P frames encoding and be capable of generating bitreams of 1, 2, or 3 temporally-independent layers.

C.7.9 Long Term Reference Frame Support

If your H.265 encoder implements Long Term Reference frames, then the following CodecAPI interfaces should be implemented (completed within one frame delay):

CODECAPI_AVEncVideoLTRBufferControl

CODECAPI_AVEncVideoMarkLTRFrame

CODECAPI_AVEncVideoUseLTRFrame

C.8 Asynchronous MFT Support

It is recommended that you implement an asynchronous MFT. An asynchronous MFT requires that you implement the IMFMediaEventGenerator and IMFShutdown interfaces.

C.9 IMFRealTimeClientEx

Registration with MMCS via IMFRealTimeCLientEx::SetWorkQueue is critical to meet performance goals; therefore, the implementation of the IMFRealTimeClientEx interface is recommended.

C.9 Multiple Instance Requirements

It is required that your H.265 encoder must support a minimum of 3 concurrent instances. The encoder should not put any limitation on the number of instances. It should be bounded by the memory usage.

These instances may be in the same process or in different processes.

C.10 Merit Validation

It is required that your H.265 encoder supports the trusted merit verification process.

It is required that your H.265 encoder be a separate binary, individually certified and signed.

C.11 Additional Requirements

Your H.265 encoder must work with the Windows MP4 file sink.

Your H.265 encoder must implement proper order of encoding configuration.

Your H.265 encoder must be capable of producing valid bitstreams with up to 3 temporal layers.

Your H.265 encoder prefix NALU prepends each slice and is syntax correct with bitstreams with 1-3 temporal layers.

Your H.265 encoder must send one batched GPU request per frame.

Your H.265 encoder must insert an AUD NALU before each frame.

Your H.265 encoder must work correctly in session 0.

C.12 Installation

It is required that your H.265 encoder is registered and unregistered along with the device driver used in the encoder.

C.113 Dynamic Format Change

It is required that your H.265 encoder supports dynamic format changes of the following within:

Profile Change

Resolution and Frame Rate Change

Add and Delete Temporal Layers

Maximum allowed number of reference frames

When multiple changes are requested before an output frame is produced, the last change should be honored.

Send comments about this topic to Microsoft

Appendix A: Removed Requirements

Appendix A: Removed requirements

The following table lists requirements that were part of the Windows 7, Windows 8, and Windows 8.1 certification programs. These requirements are not going to be part of the final Windows 10 Compatibility program.

- Device.Audio.APO.WinPEConformance
- Device.Audio.AudioController.HDControllerCompliance
- Device.Audio.Base.BasicDataFormats
- Device.Audio.Base.ChannelMasks
- Device.Audio.Base.DCOffset
- Device.Audio.Base.ExposedAudioEndpointsAreFunctional
- Device.Audio.Base.JackConnectorStateDescription
- Device.Audio.Base.JackDetection
- Device.Audio.Base.KSPROPERTYAUDIOVOLUMELEVEL
- Device.Audio.Base.KSTopologyCompliance
- Device.Audio.Base.NoUncontrollableStreamRouting
- Device.Audio.Base.NoUndiscoverableDevice
- Device.Audio.Base.RealtimeDriversSupportStandardLoopedStreaming

- Device.Audio.Base.ReportSupportedProperties
- Device.Audio.Base.TimeSynchronizedSampleRates
- Device.Audio.Base.TipRing
- Device.Audio.Base.TwoDMAEnginesAndConnections
- Device.Audio.Base.WAVEFORMATEXTENSIBLESupport
- Device.Audio.Base.WaveRTConformance
- Device.Audio.Base.ZeroGlitch
- Device.Audio.Bluetooth.AtleastOneProfileSupport
- Device.Audio.Bluetooth.DriverReqs
- Device.Audio.Bluetooth.HCIDisconnect
- Device.Audio.HardwareAudioProcessing.ETWEvent
- Device.Audio.HDAudio.HDAudioCodecAdditionalReqs
- Device.Audio.HDAudio.HDMIDCN
- Device.Audio.HDAudio.INFHasDeviceID
- Device.Audio.UAACompliance.UAA
- Device.BusController.Bluetooth.Base.RadioScanIntervalSettings
- Device.BusController.NearFieldProximity.NFCCertification
- Device.BusController.NearFieldProximity.NFCControllerNCICompliance
- Device.BusController.NearFieldProximity.ProviderImplementation
- Device.BusController.NearFieldProximity.ProximityReliability
- Device.BusController.NearFieldProximity.RangeOfActuation
- Device.BusController.NearFieldProximity.SessionEstablishmentPerformance
- Device.BusController.NearFieldProximity.TaptoSetupScenario
- Device.BusController.NearFieldProximity.TapToUseScenarios
- Device.BusController.UsbController.SpecificationCompliance
- Device.BusController.UsbController.SuperSpeedConnectorsSupportHighFullLow
- Device.BusController.UsbController.SupportSelectiveSuspend
- Device.Connectivity.BluetoothDevices.BluetoothDeviceIdProfileVer12
- Device.Connectivity.BluetoothDevices.BluetoothUSBPlugandPlay
- Device.Connectivity.BluetoothDevices.FunctionAfterSystemSuspendCycle
- Device.Connectivity.BluetoothDevices.RespondToServiceDiscoveryRequests

- Device.Connectivity.NearFieldProximity.DeviceNFCCertification
- Device.Connectivity.NearFieldProximity.DeviceRangeOfActuation
- Device.Connectivity.NearFieldProximity.DeviceTapToSetup
- Device.Connectivity.NearFieldProximity.NfcForumTag
- Device.Connectivity.NearFieldProximity.TouchMark
- Device.Connectivity.UsbDevices.Addressing
- Device.Connectivity.UsbDevices.AlternateDriver
- Device.Connectivity.UsbDevices.CompliesWithChap9
- Device.Connectivity.UsbDevices.EsdRecovery
- Device.Connectivity.UsbDevices.InstallViaUniquePnpIdentifier
- Device.Connectivity.UsbDevices.MustEnumerateOnEhciAndXhci
- Device.Connectivity.UsbDevices.MustSupportSuspendOnRT
- Device.Connectivity.UsbDevices.PeripheralOperatesInFunctionMode
- Device.Connectivity.UsbDevices.PortMove500ms
- Device.Connectivity.UsbDevices.Usb3CompatibleWithDownLevel
- Device.Connectivity.UsbHub.CompliesWithChap11
- Device.Connectivity.UsbHub.ImplementSuperSpeedDescriptors
- Device.Connectivity.UsbHub.MapPortsPerUsb3Specification
- Device.Connectivity.UsbHub.ProvideStandardInterfacesToHostPeripherals
- Device.Connectivity.UsbHub.SuperSpeedRemainsOnAfterPortReset
- Device.Connectivity.WSD.MetadataValid
- Device.DevFund.Color.DeviceColorProfilesInstall
- Device.DevFund.DriverFramework.AllDrivers.WDFLoadGroup
- Device.DevFund.DriverFramework.KMDF.HandleDDIFailures
- Device.DevFund.DriverFramework.KMDF.WDFRedistributables
- Device.DevFund.DriverFramework.UMDF.WDFRedistributables
- Device.DevFund.HALExtension.HAL
- Device.DevFund.HALExtension.HALSignatureAttributes
- Device.DevFund.Reliability.S3S4SleepStates
- Device.DevFund.Reliability.X64Support
- Device.Digitizer.Base.DigitizersAppearAsHID

- Device.Digitizer.Base.HighQualityDigitizerInput
- Device.Digitizer.Pen.100HzSampleRate
- Device.Digitizer.Pen.ContactAccuracy
- Device.Digitizer.Pen.HoverAccuracy
- Device.Digitizer.Pen.PenRange
- Device.Digitizer.Pen.PenResolution
- Device.Digitizer.Touch.5TouchPointMinimum
- Device.Digitizer.Touch.DigitizerConnectsOverUSBOrI2C
- Device.Digitizer.Touch.DigitizerJitter
- Device.Digitizer.Touch.ExtraInputBehavior
- Device.Digitizer.Touch.FieldFirmwareUpdatable
- Device.Digitizer.Touch.HIDCompliantFirmware
- Device.Digitizer.Touch.HighQualityTouchDigitizerInput
- Device.Digitizer.Touch.HighResolutionTimeStamp
- Device.Digitizer.Touch.InputSeparation
- Device.Digitizer.Touch.NoiseSuppression
- Device.Digitizer.Touch.PhysicalDimension
- Device.Digitizer.Touch.PhysicalInputPosition
- Device.Digitizer.Touch.PowerStates
- Device.Digitizer.Touch.ReportingRate
- Device.Digitizer.Touch.ResponseLatency
- Device.Digitizer.Touch.TouchResolution
- Device.Digitizer.Touch.ZAxisAllowance
- Device.Graphics.WDDM.Display.MediaCenterResolutionTiming
- Device.Graphics.WDDM.Display.ResetToVGA
- Device.Graphics.WDDM.Display.TVOut.Base
- Device.Graphics.WDDM.Display.TVOut.DAC
- Device.Graphics.WDDM.Display.TVOut.Encoder
- Device.Graphics.WDDM.DisplayRender.OutputProtection.Windows7
- Device.Graphics.WDDM.Render.Windows7.VideoDecoding
- Device.Graphics.WDDM11.Render.ContentProtection.ContentProtection

- Device.Graphics.WDDM13.DisplayRender.MultiPlaneOverlayVideoProcessing
- Device.Graphics.XDDM.Stability
- Device.Imaging.Printer.Base.autoConfiguration
- Device.Imaging.Printer.Base.connectivityRobustness
- Device.Imaging.Printer.Base.DocumentProperties
- Device.Imaging.Printer.Base.DriverEventFiles
- Device.Imaging.Printer.Base.driverIsolation
- Device.Imaging.Printer.Base.driverPackage
- Device.Imaging.Printer.Base.driverStability
- Device.Imaging.Printer.Base.faxModem
- Device.Imaging.Printer.Base.faxTIA592
- Device.Imaging.Printer.Base.faxV34
- Device.Imaging.Printer.Base.jobCancellation
- Device.Imaging.Printer.Base.metadata
- Device.Imaging.Printer.Base.portMonitors
- Device.Imaging.Printer.Base.PrinterExtension
- Device.Imaging.Printer.Base.printerInterfaces
- Device.Imaging.Printer.Cluster.cluster
- Device.Imaging.Printer.WSD.Rally
- Device.Imaging.Printer.WSD.WSPrint
- Device.Imaging.Scanner.Base.driverInstallation
- Device.Imaging.Scanner.Base.errorHandling
- Device.Imaging.Scanner.Base.metadata
- Device.Imaging.Scanner.Base.MFPmultiplePorts
- Device.Imaging.Scanner.Base.RawFileFormat
- Device.Imaging.Scanner.Base.scannerInterfaces
- Device.Imaging.Scanner.Base.statusMessages
- Device.Imaging.Scanner.Base.WIAArchitecture
- Device.Imaging.Scanner.DistributedScanManagement.DistributedScanManagement
- Device.Imaging.Scanner.WSD.Rally
- Device.Input.FingerPrintReader.BaseLegacy

- Device.Input.FingerPrintReader.ManagementAppsLegacy
- Device.Input.FingerPrintReader.SensorEngineDBLegacy
- Device.Input.FingerPrintReader.WBDILegacy

Send comments about this topic to Microsoft

Systems

In this section

- System.Client.BluetoothController.Base
- System.Client.BluetoothController.NonUSB
- System.Client.BluetoothController.USB
- System.Client.BrightnessControls
- System.Client.Camera
- System.Client.Digitizer
- System.Client.Digitizer.Touch
- System.Client.Firmware.UEFI.GOP
- System.Client.Graphics
- System.Client.MobileBroadBand
- System.Client.PCContainer
- System.Client.RadioManagement
- System.Client.RadioManagement.ConnectedStandby
- System.Client.ScreenRotation
- System.Client.SystemConfiguration
- System.Client.SystemImage
- System.Client.SystemPartition
- System.Client.Tablet.Graphics
- System.Client.WLAN.BasicConnectivity
- System.Client.WLAN.HangDetectionAndRecovery
- System.Client.WLAN.HostedNetwork
- System.Client.WLAN.Miracast

- System.Client.WLAN.WiFiDirect
- System.Fundamentals.DebugPort
- System.Fundamentals.DebugPort.USB
- System.Fundamentals.EnergyEstimation
- System.Fundamentals.Firmware
- System.Fundamentals.Firmware.Boot
- System.Fundamentals.Firmware.CS
- System.Fundamentals.Firmware.CS.UEFISecureBoot
- System.Fundamentals.Firmware.TPR
- System.Fundamentals.Graphics
- System.Fundamentals.Graphics.DisplayRender
- System.Fundamentals.Graphics.HybridGraphics
- System.Fundamentals.Graphics.InternalDisplay
- System.Fundamentals.Graphics.MultipleDevice
- System.Fundamentals.Graphics.RenderOnly
- System.Fundamentals.HAL
- System.Fundamentals.Input
- System.Fundamentals.MarkerFile
- System.Fundamentals.Network
- System.Fundamentals.NX
- System.Fundamentals.PowerManagement
- System.Fundamentals.PowerManagement.CS
- System.Fundamentals.PXE
- System.Fundamentals.Reliability
- System.Fundamentals.Security
- System.Fundamentals.SignedDrivers
- System.Fundamentals.SMBIOS
- System.Fundamentals.StorageAndBoot
- System.Fundamentals.SystemAudio
- System.Fundamentals.SystemPCIController
- System.Fundamentals.SystemUSB

Send comments about this topic to Microsoft

System.Client.BluetoothController.Base

These requirements apply to systems that have generic Bluetooth controllers

In this topic:

- System.Client.BluetoothController.Base.LEWhiteList
- System.Client.BluetoothController.Base.NoBluetoothLEFilterDriver
- System.Client.BluetoothController.Base.OnOffStateControllableViaSoftware
- System.Client.BluetoothController.Base.SimultaneousBrEdrAndLeTraffic
- System.Client.BluetoothController.Base.WLANBTCoexistence

System.Client.BluetoothController.Base.4LeSpecification

If a system includes a Bluetooth enabled controller it must support the Bluetooth 4.0 specification requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth enabled controller must comply with the Basic Rate (BR) and Low Energy (LE) Combined Core Configuration Controller Parts and Host/Controller Interface (HCI) Core Configuration requirements outlined in the Compliance Bluetooth Version 4.0 specifications.

System.Client.BluetoothController.Base.CS

Systems that support Connected Standby with Bluetooth enabled controllers must ship with Microsoft's inbox Bluetooth stack.

Applies to	Windows 10 x64 Windows 10 x86

Description

Systems that support Connected Standby that ship with Bluetooth enabled controllers must ship with Microsoft's inbox Bluetooth stack.

System.Client.BluetoothController.Base.HciExtensions [if implemented]

MSFT Defined HCI extensions support for hardware offload of advertisement and RSSI monitoring

Applies to	Windows 10 x64 Windows 10 x86

Description

Radios that support the Microsoft-OSG Defined Bluetooth HCI Extensions must comply with the specification and pass the related HLK tests. The details of the specifications will be shared at a later date. Partners will be notified via Connect.

System.Client.BluetoothController.Base.LEStateCombinations

Systems with Bluetooth enabled controllers must support a minimum set of LE state combinations.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth enabled controller must allow the spec LE state combinations (as allowed in section [Vol 6] Part B, Section 1.1.1 of the Bluetooth version 4.0 spec).

System.Client.BluetoothController.Base.LEWhiteList

Systems with Bluetooth enabled controllers must support a minimum LE allow list size of 25 entries.

Applies to	Windows 10 x64 Windows 10 x86

Description

The Bluetooth enabled controller on the system must support a minimum of 25 entries in its allow list for remote Low Energy (LE) devices.

System.Client.BluetoothController.Base.NoBluetoothLEFilterDriver

Bluetooth LE filter drivers are not allowed to load on BTHLEENUM.SYS.

Applies to	Windows 10 x64 Windows 10 x86

Description

To ensure a uniform experience across Windows Store Apps using the Bluetooth LE (GATT) WinRT API, filter drivers shall not be loaded on BTHLEENUM.SYS.

System.Client.BluetoothController.Base.OnOffStateControllableViaSoftware

Bluetooth enabled controllers' On/Off state must be controllable via software.

Applies to	Windows 10 x64

	Windows 10 x86

Description

When turning the radio “off”, Bluetooth enabled controllers shall be powered down to its lowest supported power state and no transmission/reception shall take place. Windows will terminate Bluetooth activity by unloading the inbox protocol drivers and their children, submitting the HCI_Reset command to the controller, and then setting the controller to the D3 logical power state, allowing bus drivers to power down the radio as appropriate. The radio can be completely powered off if a bus-supported method is available to turn the radio back on. No additional vendor software control components will be supported.

On turning the radio back on, the Bluetooth stack for Windows shall resume the device to D0, allowing bus drivers to restart the device. The Windows Bluetooth stack shall then reinitialize the Bluetooth enabled components of the controller.

Bluetooth Radio Management shall only be enabled for internal Bluetooth 4.0 enabled controllers.

Bluetooth enabled controllers’ On/Off state shall be controllable via software as described in Bluetooth Software Radio Switch. The Off state is defined, at a minimum, as disabling the antenna component of the Bluetooth enabled module so there can be no transmission/reception. There must not be any hardware-only switches to control power to the Bluetooth enabled radio.

The radio must maintain on/off state across sleep and reboot.

System.Client.BluetoothController.Base.SimultaneousBrEdrAndLeTraffic

Bluetooth enabled controllers must support simultaneous BR/EDR and LE traffic.

Applies to	Windows 10 x64 Windows 10 x86

Description

Bluetooth enabled controllers must allow the simultaneous use of both Basic Rate (BR)/Enhanced Data Rate (EDR) and Low Energy (LE) radios.

System.Client.BluetoothController.Base.WLANBTCoexistence

Windows Systems that support both WLAN and Bluetooth must meet WLAN-BT Co-existence requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

Windows systems that support both WLAN and Bluetooth must meet WLAN-BT Co-existence requirements listed below. The requirement is applicable to all WLAN devices across all bus types.

- Must not drop the connection with WLAN AP when Bluetooth is scanning for new devices.
- Must be able to scan simultaneously for both WLAN and Bluetooth networks.

Send comments about this topic to Microsoft

System.Client.BluetoothController.NonUSB

These requirements apply to systems that have non-USB Bluetooth enabled controllers.

In this topic:

System.Client.BluetoothController.NonUSB.NonUsbUsesMicrosoftsStack

Any platform using a non-USB connected Bluetooth enabled controller must ship with Microsoft's inbox Bluetooth stack.

Applies to	Windows 10 x64 Windows 10 x86

Description

Any platform using a non-USB connected Bluetooth enabled controller must ship with Microsoft's inbox Bluetooth stack.

System.Client.BluetoothController.NonUSB.ScoSupport

Any platform with a non-USB connected Bluetooth enabled controller must use a sideband channel for SCO.

Applies to	Windows 10 x64 Windows 10 x86

Description

Any platform using a Non-USB connected Bluetooth enabled controller must use sideband channel for SCO (such as SCO over an I2S/PCM interface).

Send comments about this topic to Microsoft

System.Client.BluetoothController.USB

These requirements apply to systems that have USB Bluetooth enabled controllers.

In this topic:

System.Client.BluetoothController.USB.ScoDataTransportLayer

Bluetooth enabled host controllers support the SCO data transport layer as specified in the Bluetooth 2.1+EDR specifications.

Applies to	Windows 10 x64 Windows 10 x86

Description

A System with a Bluetooth enabled controller must comply with the Synchronous Connection Oriented (SCO)-USB requirements that are outlined in the Specification of the Bluetooth System, Version 2.1 + Enhanced Data Rate (EDR), Part A, Section 3.5.

Send comments about this topic to Microsoft

System.Client.BrightnessControls

This section describes requirements systems with brightness controls.

In this topic:

System.Client.BrightnessControls.BacklightOptimization

Windows Display Driver Model (WDDM) 1.2 drivers must enable scenario based backlight power optimization to reduce backlight level used by integrated panel.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- If WDDM driver supports scenario based backlight power optimization, it must indicate the support by implementing the DXGK_BRIGHTNESS_INTERFACE2 interface.
- When Windows sets the current scenario by using the DxgkDdiSetBacklightOptimization function, the WDDM driver is required to honor the intent of the scenario as follows:
  - DxgkBacklightOptimizationDisable: Driver is required to completely disable all backlight optimization.
  - DxgkBacklightOptimizationDesktop: Driver is required to enable backlight optimization at a lower aggressiveness level. Driver must optimize for scenarios like photo viewing, browser, and Office documents.
  - DxgkBacklightOptimizationDynamic: Driver is required to enable backlight optimization at a higher aggressiveness level. Driver must optimize for scenarios like video playback and gaming.
  - DxgkBacklightOptimizationDimmed: Driver is required to enable backlight optimization at a higher aggressiveness level. Driver must make sure that the content on the screen is visible but it need not be easily readable.
- Driver is allowed to dynamically change the aggressiveness level based on the content on the screen.
- Driver is required to handle Windows requests for change to brightness level (based on user input or ambient light sensor) while keeping backlight optimization enabled.
- Driver is required to gradually transition between aggressiveness levels:
  - This is important in the case when user briefly invokes playback controls. At that time, Windows will reset the scenario from DxgkBacklightOptimizationDynamic to DxgkBacklightOptimizationDesktop. The transition must not be a step but must be gradual.
- WDDM driver is required to provide accurate information when Windows queries DxgkDdiGetBacklightReduction.

- Connecting additional display devices to the system must not impact the ability to perform backlight optimization on the integrated panel of the system.

System.Client.BrightnessControls.BrightnessControlButtons

Systems that have built in physical brightness control function keys use standard ACPI events and support control of LCD backlight brightness via ACPI methods in the system firmware.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows provides users with an LCD brightness control user interface. If the system implements keys that are invisible to the operating system, these keys must use Advanced Configuration and Power Interface (ACPI) methods. These keys must not directly control the brightness after Bit 2 of the _DOS method has been set. This requires the implementation of ACPI brightness methods in the system firmware.

The following methods are required:

_BCL

_BCM

Bit 2 of the _DOS method must be disabled so that the system firmware cannot change the brightness levels automatically.

The following methods are optional:

Support for the _BQC method is highly recommended but not required. Systems must map keys to the following ACPI notification values:

- ACPI_NOTIFY_CYCLE_BRIGHTNESS_HOTKEY 0x85
- ACPI_NOTIFY_INC_BRIGHTNESS_HOTKEY 0x86
- ACPI_NOTIFY_DEC_BRIGHTNESS_HOTKEY 0x87
- ACPI_NOTIFY_ZERO_BRIGHTNESS_HOTKEY 0x88

Design Notes:

The _BCL and _BCM methods in the firmware enable the operating system to query the brightness range and values and to set new values. Refer to the ACPI 3.0 specification for more details.

System.Client.BrightnessControls.SmoothBrightness

Driver must support a smooth transition in response to all brightness change requests from Windows.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- All Windows systems that support brightness control, are required to support smooth brightness control
- All Windows systems are required to report 101 brightness levels to Windows. Brightness is reported as a % so this means 0 to 100 levels, including 0 and 100. Internally the driver might support more granular brightness control.
- This is to ensure that Windows has the ability to make fine grained changes to the screen brightness. However, the brightness slider UI might expose fewer levels through the slider because it might be cumbersome for the user to adjust so many levels.
- WDDM driver is required to implement smooth brightness control in the driver without depending on the embedded controller (EC) for the smoothness.
- WDDM driver is required to indicate support for smooth brightness control using the capability bit defined in the DXGK_BRIGHTNESS_INTERFACE2 interface.
- WDDM driver must enable/disable smooth brightness control based on state set using DxgkDdiSetBrightnessState.
- When Windows requests a change to brightness, driver is required to gradually change the brightness level over time so that the change is not a step.
- WDDM driver is allowed to select an appropriate slope for transition. However, the transition must complete in less than 2s.
- WDDM driver is allowed to alter the slope based on panel characteristics to ensure smoothness of brightness control.
- WDDM driver is required to start responding immediately to new brightness level requests. This must be honored even if the system is already in the process of an existing transition. At such a time, the system must stop the existing transition at the current level and start the

new transition from the current position. This will ensure that when a user is using the slider to manually adjust the brightness, the brightness control is still responsive and not sluggish.

- WDDM driver is required to continue supporting smooth brightness control, even if content based adaptive brightness optimization is currently in effect.
- When WDDM driver is pnp started, it must detect the brightness level applied by the firmware and smoothly transition from that level to the level set by Windows.
- Connecting additional display devices to the system must not impact the ability to do smooth brightness control on the integrated panel of the system.
- Brightness levels are represented as a % in Windows. Therefore there is no absolute mapping between brightness % level and physical brightness level. For Windows 8, the following is the guidance.

Percent represented to Windows	User Experience
0%	Brightness level such that the contents of the screen are barely visible to the user
100%	Max brightness supported by panel

Send comments about this topic to Microsoft

System.Client.Camera

In this topic:

System.Client.Camera.Device

Systems with integrated cameras must meet camera device requirements.

Applies to	Windows 10 x64

	Windows 10 x86

Description

Each integrated camera on a system must comply with **Device.Streaming.Camera.Base** and all related requirements. If the integrated camera is a USB camera, it must also comply with **Device.Streaming.Camera.UVC** for the system seeking certification.

Note: With regards to '**Device.Streaming.Camera.Base.UsageIndicator'** if a system has multiple cameras, then one physical indicator (i.e. LED) is acceptable so long as all cameras can control it. Systems without a display must have a physical indicator.

System.Client.Camera.PhysicalLocation

Systems with integrated cameras must report the physical location of each camera.

Applies to	Windows 10 x64 Windows 10 x86

Description

For any camera device that is built into the chassis of the system and has mechanically fixed direction, the firmware must provide the _PLD method and set the panel field (bits [69:67]) to the appropriate value for the panel on which the camera is mounted. For example, "Front" indicates the camera faces the user, while "back" indicates that the camera faces away from the end user.

In addition, bit 143:128 (Vertical Offset), and bits 159:144 (Horizontal Offset) must provide the relative location of the camera with respect to the display. This origin is relative to the native pixel addressing in the display component. The origin is the lower left hand corner of the display, where positive Horizontal and Vertical Offset values are to the right and up, respectively. For more information see the ACPI version 5.0 Section 6.1.8 "Device Configuration _PLD (Physical Device Location)."

All other fields in the _PLD are optional.

System.Client.Camera.VideoCaptureAndCameraControls

Systems with integrated cameras meet the requirements of, and can support the Windows Capture Infrastructure.

Applies to	Windows 10 x64 Windows 10 x86

Description

System Memory:

System Memory must be supported.

Independent Streaming:

All integrated Cameras must support independent streaming between different pins and different filters (cameras) according to the capabilities listed in the Profiles advertised by the device. If the camera does not support Profiles, then concurrent streaming for all system cameras is optional.

Mirroring:

The default state for mirroring must be "not mirrored."

Camera Controls (If Implemented):

Each of the following camera controls are optional.

- Region of Interest (ROI)
- Focus
- Exposure
- White Balance
- Zoom
- Camera Flash
- Scene Mode
- Optimization Hints
- Optical Image Stabilization
- Backlight Compensation
- Brightness
- Contrast
- Exposure Compensation
- Hue
- Pan
- Tilt
- Roll

If any individual control is implemented in the camera driver, it must comply with the control specification in the WDK.

Photo Sequence (If Implemented):

Photo Sequence captures a sequence of photos in response to a single photo click. Capture pipeline would send buffers to the camera driver continuously to capture the photos in sequence. This mode also allows capturing photos from the time before the “user click” thus helping users not to lose a moment.

If camera HW supports Photo Sequence, it must expose the capability through the Photo Mode property and comply with the performance requirements.

Photo Sequence must be enabled by the device and driver to:

- Support the same resolutions that are exposed in Normal mode
- Report the current frame rates possible in Photo Sequence Mode based on the current light conditions. Device must honor and not exceed the maximum frame rate set by the application.
- Support at minimum 4fps measured at lesser of the maximum resolution exposed by the image pin or 8MP.
- Provide at least 4 frames in the past at lesser of the maximum resolution exposed by the image pin or 8MP.
- Photo Sequence should be performed independently, regardless preview on/off.
- Provide frames continuously in Photo Sequence mode at lesser of the maximum resolution exposed by the image pin or 8MP.
- If the driver outputs JPEG format for Photo Sequence it must also support thumbnails, upon request, at 1/2x, 1/4x, 1/8x, and 1/16x of the width and height of the original image resolution.
- The JPEG image generated by the camera may optionally have EXIF metadata indicating the “flash fired” information. EXIF information shall not include personally identifiable information, such as location, unique ids, among others.

Variable Photo Sequence (If Implemented):

Variable Photo Sequence captures a finite number of images and supports the ability to vary the capture parameters for each of the captured images. If implemented in Camera driver then the driver should be able to return the requested number of images, in order, each with varying capture parameters as instructed by the application. The driver shall be able to preprogram the number of frames needed and set independent capture parameters for each frame before capture is initiated.

It is recommended that the variable photo sequence allows the application to specify the following parameters for each frame, but at least one of these must be implemented if VPS is supported:

- Exposure
- ISO

- Exposure Compensation in EV
- Focus position
- Flash

If any parameter is not set in per frame settings the driver shall follow the global settings and 3A locks. For example when EV bracketing is used, the driver shall ensure that exposure related parameters like gain and exposure are set according to EV bracketing settings. The driver may vary auto white balance settings for image frames unless the per frame settings use manual white balance settings or in case of application uses white balance lock. It not recommended that lens position is automatically changed between the VPS frames (unless manually specified by the application).

Send comments about this topic to Microsoft

System.Client.Digitizer

In this topic:

System.Client.Digitizer.Base.SystemDigitizerBase

System Digitizer Base

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following Digitizer Base device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Base** requirements for full requirement details:

Device.Input.Digitizer.Base.ContactReports

Device.Input.Digitizer.Base.HIDCompliant

Device.Input.Digitizer.Base.ThirdPartyDrivers

System.Client.Digitizer.SystemTouch

System Touch

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch** requirements for full requirement details:

Device.Input.Digitizer.Touch.Accuracy

Device.Input.Digitizer.Touch.Buffering

Device.Input.Digitizer.Touch.CustomGestures

Device.Input.Digitizer.Touch.FingerSeparation

Device.Input.Digitizer.Touch.Jitter

Device.Input.Digitizer.Touch.Latency

Device.Input.Digitizer.Touch.MinContactCount

Device.Input.Digitizer.Touch.ReportRate

Device.Input.Digitizer.Touch.Resolution

Microsoft strongly recommends touch solutions capable of reporting 5 or more simultaneous contact points. This ensures that the platform is compatible with third party applications that rely upon touch input, and that end users are able to invoke all of the system gestures provided by Windows.

Microsoft recognizes that extenuating circumstances exist whereby an extended gesture experience is not necessary. In order to accommodate this very limited set of systems, we make the following allowances:

Systems that are sold as build to configure, custom enterprise images, or are designed for specific vertical enterprise markets, have the option to ship a touch screen capable of reporting only a single contact point. Examples include systems designed for health care, military applications, and Point of Sale.

Any system incapable of supporting more than a single contact point will be unable to invoke any system gestures other than generic mouse-like behavior.

A system reliant upon a keyboard and mouse as the primary input modality, without the capability to convert into a tablet mode device, may choose to integrate a touch solution capable of supporting a minimum of 2 simultaneous contact points. Examples include: external displays, All-In-One desktop systems.

Any system incapable of supporting more than 2 simultaneous contact points will be unable to invoke 4 finger accessibility gestures.

All other systems must support a minimum of 5 simultaneous contact points

System.Client.Digitizer.SystemPrecisionTouchpad

Precision Touchpad

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following Precision Touchpad device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.PrecisionTouchpad** requirements for full requirement details:

Device.Input.Digitizer.PrecisionTouchpad.Accuracy

Device.Input.Digitizer.PrecisionTouchpad.Buffering

Device.Input.Digitizer.PrecisionTouchpad.Buttons

Device.Input.Digitizer.PrecisionTouchpad.ContactTipSwitchHeight

Device.Input.Digitizer.PrecisionTouchpad.DeviceTypeReporting

Device.Input.Digitizer.PrecisionTouchpad.Dimensions

Device.Input.Digitizer.PrecisionTouchpad.FingerSeparation

Device.Input.Digitizer.PrecisionTouchpad.Jitter

Device.Input.Digitizer.PrecisionTouchpad.Latency

Device.Input.Digitizer.PrecisionTouchpad.MinMaxContacts

Device.Input.PrecisionTouchpad.Precision.InputResolution

Device.Input.Digitizer.PrecisionTouchpad.SelectiveReporting

A touchpad may not be marketed as a Precision Touchpad if the device requires a 3rd party driver be installed in order to report as a Precision Touchpad.

Send comments about this topic to Microsoft

System.Client.Digitizer.Touch

In this topic:

- System.Client.Digitizer.Touch.HIDCompliant
- System.Client.Digitizer.Touch.Jitter
- System.Client.Digitizer.Touch.Latency
- System.Client.Digitizer.Touch.MinContactCount
- System.Client.Digitizer.Touch.ReportRate
- System.Client.Digitizer.Touch.Resolution
- System.Client.Digitizer.Touch.ThirdPartyDrivers

System.Client.Digitizer.Touch.Accuracy

System Touch Accuracy

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.Accuracv** requirement for full requirement details.

System.Client.Digitizer.Touch.Buffering

System Touch Buffering for Buses with High Resume Latency

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.Buffering** requirement for full requirement details.

System.Client.Digitizer.Touch.ContactReports

System Touch Digitizer Reliability

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.ContactReports** requirement for full requirement details.

System.Client.Digitizer.Touch.CustomGestures

System Touch Custom Run-Time System Gestures

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.CustomGestures** requirement for full requirement details.

System.Client.Digitizer.Touch.FingerSeparation

System Touch Finger Separation

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.FingerSeparation** requirement for full requirement details.

System.Client.Digitizer.Touch.HIDCompliant

System Touch HID Compliant Device Firmware and/or HID Mini-port Driver

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.HIDCompliant** requirement for full requirement details.

System.Client.Digitizer.Touch.Jitter

System Touch Jitter and Linearity

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.Jitter** requirement for full requirement details.

System.Client.Digitizer.Touch.Latency

System Touch Response Latency

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.Latency** requirement for full requirement details.

System.Client.Digitizer.Touch.MinContactCount

System Touch Minimum Simultaneous Reportable Contacts

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.MinContactCount** requirement for full requirement details.

System.Client.Digitizer.Touch.ReportRate

System Touch Report Rate

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.ReportRate** requirement for full requirement details.

System.Client.Digitizer.Touch.Resolution

System Touch Input Resolution

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.Resolution** requirement for full requirement details.

System.Client.Digitizer.Touch.ThirdPartyDrivers

System Touch Servicing and 3rd Party Driver Availability

Applies to	Windows 10 x64 Windows 10 x86

Description

The following Touch device level requirements must be met and verified upon integration into a system. Please refer to the following **Device.Input.Digitizer.Touch.ThirdPartyDrivers** requirement for full requirement details.

Send comments about this topic to Microsoft

System.Client.Firmware.UEFI.GOP

In this topic:

System.Client.Firmware.UEFI.GOP.Display

System firmware must support Graphics Output Protocol (GOP) and Windows display requirements.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Every firmware on a Windows client system must support the GOP as defined in UEFI 2.3.1.

The display is controlled by the system UEFI before the WDDM graphics driver takes over. GOP must be available when the Windows EFI boot manager loads. VBIOS is not supported. It is also required for prior UI, such as OEM logo, firmware setup, or password prompt screens to enable GOP. During this time when the firmware is in control, the following are the requirements.

Topology Selection

- UEFI must reliably detect all the displays that are connected to the POST adapter. The Pre-OS screen can only be displayed on a display connected to the POST adapter.
- In case multiple displays are detected, UEFI must display the Pre-OS screen based on the following logic:
  - System with an Integrated display(Laptop, All In One, Tablet): UEFI must display the Pre-OS screen only on the integrated display.
  - System without an Integrated display (integrated display is shut or desktop system): UEFI must display the Pre-OS screen on one display. UEFI must select the display by prioritizing the displays based on connector type. The prioritization is as follows: DisplayPort, HDMI, DVI, HD15, Component, S-Video. If there are multiple monitors connected using the same connector type, the firmware can select which one to use.

Mode Selection

- Once UEFI has determined which display to enabled to display the Pre-OS screen, it must select the mode to apply based on the following logic.
  - System with an Integrated display (Laptop, All In One, Tablet): The display must always be set to its native resolution and native timing.
  - System without an Integrated display (desktop):
    - UEFI must attempt to set the native resolution and timing of the display by obtaining it from the EDID.
    - If that is not supported, UEFI must select an alternate mode that matches the same aspect ratio as the native resolution of the display.

- At the minimum, UEFI must set a mode of 1024 x 768.
- If the display device does not provide an EDID, UEFI must set a mode of 1024 x 768.

- The firmware must always use a 32 bit linear frame buffer to display the Pre-OS screen.
- PixelsPerScanLine must be equal to the HorizontalResolution.
- PixelFormat must be PixelBlueGreenRedReserved8BitPerColor. Note that a physical frame buffer is required; PixelBltOnly is not supported.

Mode Pruning

- UEFI must prune the list of available modes in accordance with the requirements called out in EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode() (as specified in the UEFI specification version 2.1)

Providing the EDID

- Once the UEFI has set a mode on the appropriate display (based on Topology Selection), UEFI must obtain the EDID of the display and pass it to Windows when Windows uses the EFI_EDID_DISCOVERED_PROTOCOL (as specified in the UEFI specification version 2.1)to query for the EDID:
  - It is possible that some integrated panels might not have an EDID in the display panel itself. In this case, UEFI must manufacture the EDID. The EDID must accurately specify the native timing and the physical dimensions of the integrated panel.
  - If the display is not integrated and does not have an EDID, then the UEFI does not need to manufacture an EDID.

Send comments about this topic to Microsoft

System.Client.Graphics

In this topic:

- System.Client.Graphics.WirelessUSBDisplay

System.Client.Graphics.FullGPU

A Windows client system must have a "Full" graphics device and that device must be the post device.

Applies to	Windows 10 x64 Windows 10 x86

Description

WDDM 1.3 introduces multiple driver/device types: Full, Render only, and Display only. For a detailed description of each, refer to the WDDM 1.3 in requirement **Device.Graphics.WDDM13.Base**.

Each of these driver/device types are designed for specific scenarios and usage case. All client scenarios expect a "full" graphics device. Also many applications assume that the post device is the "best" graphics devices and use that device exclusively. For this reason, a Windows client system must have a "full" graphics driver/device that is capable of display, rendering, and video.

System.Client.Graphics.NoMoreThanOneInternalMonitor

Graphics driver must not enumerate more than one monitor as internal.

Applies to	Windows 10 x64 Windows 10 x86

Description

The graphics driver must not enumerate more than one display target of the D3DKMDT_VOT_INTERNAL type on any adapter.

System.Client.Graphics.WDDM

All Windows graphics drivers must be Windows Display Driver Model (WDDM).

Applies to	Windows 10 x64 Windows 10 x86

Description

The WDDM architecture offers functionality to enable features such as desktop composition, enhanced fault Tolerance, video memory manager, scheduler, cross process sharing of D3D surfaces and so on. WDDM was specifically designed for modern graphics devices that are a minimum of Direct3D 10 Feature Level 9_3 with pixel shader 2.0 or better and have all the necessary hardware features to support the WDDM functionality of memory management, scheduling, and fault tolerance.

WDDMv1.3 is required by all systems shipped with Windows 10.

Table below explains the scenario usage for the Graphic driver types:

	Client	Server	Client running in a Virtual Environment	Server Virtual
Full Graphics	Required as post device	Optional	Optional	Optional
Display Only	Not allowed	Optional	Optional	Optional
Render Only	Optional as non primary adapter	Optional	Optional	Optional
Headless	Not allowed	Optional	N/A	N/A

System.Client.Graphics.WDDMSupportRotatedModes

If accelerometer is present, Windows Display Driver Model (WDDM) driver must support all rotated modes.

Applies to	Windows 10 x64 Windows 10 x86

Description

On a system with an accelerometer, the WDDM driver is required to support all rotated modes for every resolution enumerated for the integrated panel:

- A WDDM driver is required to enumerate source modes for the integrated display. The WDDM driver must support rotated modes (0, 90, 180 and 270) for every mode that it enumerates for the integrated panel.
- The rotation is required to be supported even if the integrated panel is in a duplicate or extended topology with another display device. For duplicate mode, it is acceptable to rotate all targets connected to the rotated source. Per path rotation is allowed but not required.

Both the above mentioned requirements are optional for Stereo 3D capable resolutions.

System.Client.Graphics.WirelessUSBDisplay

System limitations for wireless and USB connected displays.

Applies to	Windows 10 x64 Windows 10 x86

Description

- Display devices (Monitor, LCD, TV, Projectors) are enumerated to Windows only via the WDDM Graphics driver.
- There must be at least one display device physically connected to a full WDDM graphics hardware that supports at least DX 9_1 in the hardware.
- Windows only supports a fixed set of display connectors as defined in WDDM as part of the D3DKMDT_VIDEO_OUTPUT_TECHNOLOGY enumeration.
- The WDDM driver is required to accurately report the connection medium used to connect the display device to the system.
- Windows supports wireless displays via Miracast connection only, via WDDM1.3 or WDDM2.0 display drivers.
- Systems may connect a display using the USB Type C alternate mode for DisplayPort, and this display should be enumerated as a typical DisplayPort connection.
- No other types of wireless display or USB display are supported.

Send comments about this topic to Microsoft

System.Client.MobileBroadBand

These are requirements for Mobile Broadband devices integrated in the systems.

In this topic:

System.Client.MobileBroadBand.ClassDriver

USB interface based GSM and CDMA class of Mobile Broadband device firmware must comply with USB-IF's Mobile Broadband Interface Model Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB interface based GSM and CDMA class of Mobile Broadband device firmware implementation must comply with the USB-IF's Mobile Broadband Interface Model (MBIM) Specification. No additional IHV drivers are needed for the functionality of the device and the device must work with Microsoft's Mobile Broadband(MB) class driver implementation. Note that Microsoft generic class driver doesn't support non-USB interface devices. Non-USB based devices require device manufacturer's device driver compliant with MB driver model specification.

Additional Details: Mobile Broadband Interface Model Specification:

http://www.usb.org/developers/devclass_docs/MBIM10.zip Mobile Broadband Driver Model Specification: http://msdn.microsoft.com/en-us/library/windows/hardware/ff560543(v=VS.85).aspx

Exception: - Device models that are announced as End of life (EOL) as of December, 2011.- Device models that are no longer in production line. Note that above exceptions are applicable only if:- devices are used in Windows 8 Client x86 and Windows 8 Client x64. - devices are pre-certified for multiple operators (at least 20).

System.Client.MobileBroadBand.ConcurrentRadioUsage

System Builders must ensure that the RF performance is optimized for Mobile Broadband, Wi-Fi and Bluetooth enabled radios running at the same time.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

System Builders must ensure that the RF performance is optimized for Mobile Broadband, Wi-Fi and Bluetooth enabled radios running at the same time. Systems that enable internet connection sharing (tethering), multi-homing, and network switching all require multiple radios to be active simultaneously. Systems should ensure high throughput, high reliability, optimal power efficiency and minimum RF interference under these conditions regardless of the system form factor.

System.Client.MobileBroadBand.MobileBroadBand

Systems that include Broadband support meet Windows requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Firmware Requirements

USB based devices for GSM and CDMA technologies (3GPP/3GPP2 standards based) need to be firmware

compliant with the Mobile Broadband Interface Model specification. These devices need to be certified by the

USB Forum for compliance (when it becomes available for MB devices).

In addition to the above, firmware needs to support the features listed below as specified by NDIS.

Firmware Feature	Requirement
No Pause on Suspend	Required
USB Selective Suspend	Required – If USB based
Radio Management	Required
Wake on Mobile Broadband	Required
Fast Dormancy	Required

No additional Connection Manager software is required for the operation of mobile broadband devices.

Value-add Mobile Broadband Connection Managers, if implemented, need to implement the Mobile Broadband

API (http://msdn.microsoft.com/en-us/library/dd323271(VS.85).aspx).

Microsoft strongly recommends USB-based bus interfaces such as analog USB, HSIC (where applicable) and SSIC

(When available). Mobile Broadband stack in Windows 8 is designed to support only USB protocol based bus

interfaces. The following table summarizes the required mobile broadband features.

Attribute	Requirement
Bus	USB-HSCI (preferred) or USB

Systems must also comply with Mobile Broadband requirements:

- Devices MUST support 16 bitmap wake patterns of 128 bytes each.
- Devices MUST wake the system on register state change.
- Devices MUST wake the system on media connect.
- Devices MUST wake the system on media disconnect.
- GSM and CDMA class of Devices MUST wake the system on receiving an incoming SMS message.
- Devices that support USSD MUST wake the system on receiving USSD message.
- Devices MUST support wake packet indication. NIC should cache the packet causing the wake on hardware and pass it up when the OS is ready for receives.
- Mobile Broadband class of devices must support Wake on Mobile Broadband. It should wake the system on above mentioned events. Note that wake on USSD is mandatory only if the device reports that it supports USSD. Else it is optional. See the following MSDN documentation for more information on the SMS and register state wake events.
- NDIS_STATUS_WWAN_REGISTER_STATE
- NDIS_STATUS_WWAN_SMS_RECEIVE

Send comments about this topic to Microsoft

System.Client.PCContainer

Windows is moving towards a device centric presentation of computers and devices. Elements of the Windows user interface (UI), such as the Devices and Printers folder, will show the computer and all devices that are connected to the computer. The requirements in this section detail what is required to have the PC appear as a single object in the Windows UI.

In this topic:

System.Client.PCContainer.PCAppearsAsSingleObject

Computers must appear as a single object in the Devices and Printers folder.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Computers must appear as a single object in the Devices and Printers folder. Windows has a platform layer which groups all functionality exposed by the computer into a single object. This object is referred to as the computer device container. The computer device container must contain all of the device functions that are located physically inside the computer chassis. This includes, but is not limited to, keyboards, touch-pads; media control/media transport keys, wireless radios, storage devices, and audio devices. The computer device container is used throughout the Windows platform and is visibly exposed to the user in the Devices and Printers user interface. This requirement ensures a consistent and high quality user experience by enforcing the "one object per physical device" rule in the Devices and Printers folder.

The computer must appear as a single device container in the Devices and Printers folder for the following reason:

- Devices and Printers will be unable to provide a logical and understandable representation of the computer to the user. Accurate information as to which devices are physically integrated with the computer must be supplied to support this and dependent Windows features.

Design Notes:

Windows is moving towards a device centric presentation of computers and devices. The Devices and Printers folder will show the computer and all devices that are connected to the computer. In Devices and Printers the computer is represented by a single icon. All of the functionality exposed by the computer will be available through this single icon object, providing one location for users to discover devices integrated with the computer and execute specific actions on those integrated devices. To enable this experience, the computer must be able to detect and group all computer integrated devices (all devices physically inside the PC). This requires that computer integrated devices properly identify themselves as integrated components. This can be achieved by indicating that the device is not removable from computer, properly configuring ACPI for the port to which the device is attached, or creating a registry DeviceOverride entry for the device. (Note: Each bus type has different mechanisms for identifying the removable relationship for devices attached to that bus.

To group the functionality exposed by the computer into a single device container, Windows uses information available in the device hardware, bus driver, and system UEFI or BIOS and Windows

registry. The bus type to which a given device is attached determines the heuristic Windows applies to group that device. The whitepaper titled "Multifunction Device Support and Device Container Groupings in Windows 7," which can be found at http://www.microsoft.com/whdc/Device/DeviceExperience/ContainerIDs.mspx, explains the heuristic for many bus types, including:

- Universal Serial Bus (USB)
- Bluetooth
- IP connected devices using Plug and Play Extensions (PnP-X)
- 1394
- eSATA
- PCI Express (PCIe)

The Single Computer Display Object test (ComputerSingleDDOTest.exe) must be executed on the system to check if this requirement has been met. The tool is available in Windows Lab Kit.

Send comments about this topic to Microsoft

System.Client.RadioManagement

This feature contains requirements for buttons that control the management of any radios in a laptop or Tablet/convertible PC. It also contains requirements for GPS radios, Near Field Proximity radios, and Bluetooth enabled radios that do not use the Windows native Bluetooth stack.

In this topic:

System.Client.RadioManagement.HardwareButton

If a PC has a physical (hardware) button switch on a PC that turns wireless radios on and off, it must be software controllable and interact appropriately with the Radio Management UI.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

There does not need to be a hardware button for wireless radios on Windows 10 laptops or tablet/convertible PCs. A wireless hardware button is one of the following:

- Toggle Button (Laptops and Tablets)
- Toggle Button with LED (Non-Connected standby supported laptops and tablets)
- A-B slider switch (Laptops and Tablets)
- A-B slider switch with LED (Non-Connected standby supported laptops and tablets)

When there is a hardware button for wireless radios there must not be more than one, and it must control all the radios present in the computer.An LED to indicate the state of the switch is optional. Please note that an LED indicating wireless status is not allowed on systems that support connected standby. If an LED is present along with the button, it must behave as defined here:

- There must only be one LED to indicate wireless status (there must not be one LED for Bluetooth, one for Wi-Fi, etc.).
- If the global wireless state is ON, the LED must be lit.
- When the global wireless state is OFF, the LED must not be lit.
- When the button is pressed or switch is flipped, it must send a HID message that can be consumed by the Radio Management API.
- When the Radio Management API sends a HID message, the button or switch must receive the message and change the state of the LED accordingly.

System.Client.RadioManagement.RadioMaintainsState

Radio maintains on/off state across sleep and reboot power cycles.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The state of the wireless radio must persist across sleep, reboot, user log off, user switching and hibernate.

System.Client.RadioManagement.RadioManagementAPIHID

Wireless hardware button must communicate the change of state to the Radio Management API using HID.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

When the state of wireless radio switch changes, whether it is a slider A-B switch (with or without LED) or toggle button (with or without LED), this HID-compliant hardware switch/button must expose the HID collections to be consumed by the radio management API. Toggle button must not change the state of the device radio directly. A-B switch can be wired directly to the radios and change their state as long as it communicates the change of state to the Radio Management API using the HID driver and it changes the state in all radios present in the PC. The HID usage IDs are:

Usage ID	Usage Name	Usage Type
0x0C	Wireless Radio Controls	CA
0xC6	Wireless Radio Button	OOC
0xC7	Wireless Radio LED	OOC
0xC8	Wireless Radio Slider Switch	OOC

The collections are:

Button without LED (stateless button) – For laptops, tablets and convertibles

USAGE_PAGE (Generic Desktop) 05 01
USAGE (Wireless Radio Controls) 09 0C
COLLECTION (Application) A1 01
  LOGICAL_MINIMUM (0) 15 00
  LOGICAL_MAXIMUM (1) 25 01
  USAGE (Wireless Radio Button) 09 C6
  REPORT_COUNT (1) 95 01
  REPORT_SIZE (1) 75 01
  INPUT (Data,Var,Rel) 81 06

REPORT_SIZE (7)	75 07
INPUT (Cnst,Var,Abs)	81 03
END_COLLECTION	C0

Button with LED – For laptops, tablets and convertibles that do NOT support connected standby

USAGE_PAGE (Generic Desktop)	05 01
USAGE (Wireless Radio Controls)	09 0C
COLLECTION (Application)	A1 01
LOGICAL_MINIMUM (0)	15 00
LOGICAL_MAXIMUM (1)	25 01
USAGE (Wireless Radio Button)	09 C6
REPORT_COUNT (1)	95 01
REPORT_SIZE (1)	75 01
INPUT (Data,Var,Rel)	81 06
REPORT_SIZE (7)	75 07
INPUT (Cnst,Var,Abs)	81 03
USAGE (Wireless Radio LED)	09 C7
REPORT_SIZE (1)	75 01
OUTPUT (Data,Var,Rel)	91 02
REPORT_SIZE (7)	75 07
OUTPUT (Cnst,Var,Abs)	91 03
END_COLLECTION	C0

Slider Switch (without LED) - For laptops, tablets and convertibles

USAGE_PAGE (Generic Desktop)	05 01
USAGE (Wireless Radio Controls)	09 0C
COLLECTION (Application)	A1 01
LOGICAL_MINIMUM (0)	15 00
LOGICAL_MAXIMUM (1)	25 01
USAGE (Wireless Radio Slider Switch)	09 C8
REPORT_COUNT (1)	95 01
REPORT_SIZE (1)	75 01
INPUT (Data,Var,Abs)	81 02
REPORT_SIZE (7)	75 07
INPUT (Cnst,Var,Abs)	81 03
END_COLLECTION	C0

Slider Switch with LED- Laptops, tablets and convertibles that do NOT support connected standby

USAGE_PAGE (Generic Desktop)	05 01
USAGE (Wireless Radio Controls)	09 0C
COLLECTION (Application)	A1 01
LOGICAL_MINIMUM (0)	15 00
LOGICAL_MAXIMUM (1)	25 01
USAGE (Wireless Radio Slider Switch)	09 C8
REPORT_COUNT (1)	95 01
REPORT_SIZE (1)	75 01
INPUT (Data,Var,Abs)	81 02
REPORT_SIZE (7)	75 07
INPUT (Cnst,Var,Abs)	81 03
USAGE (Wireless Radio LED)	09 C7
REPORT_SIZE (1)	75 01
OUTPUT (Data,Var,Rel)	91 02
REPORT_SIZE (7)	75 07
OUTPUT (Cnst,Var,Abs)	91 03
END_COLLECTION	C0

LED Only (No button or slider) - Laptops, tablets and convertibles that do NOT support connected standby

USAGE_PAGE (Generic Desktop)	05 01
USAGE (Wireless Radio Controls)	09 0C
COLLECTION (Application)	A1 01
LOGICAL_MINIMUM (0)	15 00
LOGICAL_MAXIMUM (1)	25 01
USAGE (Wireless Radio LED)	09 C7
REPORT_COUNT (1)	95 01
REPORT_SIZE (1)	75 01
OUTPUT (Data,Var,Rel)	91 02
REPORT_SIZE (7)	75 07
OUTPUT (Cnst,Var,Abs)	91 03
END_COLLECTION	C0

Wireless radio LED must have a HID-compliant driver to reflect the state of the airplane mode switch located in the user interface. Wireless radio LED only uses HID for output (no input since there is no button).

When the Radio Management API sends a HID message because the global wireless state (airplane mode) has changed, the switch must consume this message and toggle the state.

For an A-B switch, the manufacturer's proprietary embedded controller must report the correct state of the switch at all times by sending a HID message to the HID driver, including every time the PC is turned on back on. Reporting the state of the A-B switch when the computer is turned back on is especially important in the case that the switch changed states while the PC was in states S3/S4/S5.

Send comments about this topic to Microsoft

System.Client.RadioManagement.ConnectedStandby

This feature contains requirements for buttons that control the management of any radios in a laptop or Tablet/convertible PC. The radios that this requirement applies to are GPS.

In this topic:

System.Client.RadioManagement.ConnectedStandby.NoRadioStatusIndicatorLights

Systems that support Connected Standby must not include a light indicating the status of the radios in the system.

Applies to	Windows 10 x64 Windows 10 x86

Description

In order to conserve energy, systems that support connected standby cannot include a status indicator indicating whether the radios are on.

Send comments about this topic to Microsoft

System.Client.ScreenRotation

In this topic:

System.Client.ScreenRotation.SmoothRotation

Systems with accelerometers perform screen rotation in 300 milliseconds and without any video glitches.

Applies to	Windows 10 x64 Windows 10 x86

Description

All Windows systems with an accelerometer must have sufficient graphics performance to meet performance requirements for screen rotation:

- A WDDM driver is required to enumerate source modes for the integrated display. The WDDM driver must support rotated modes (0, 90, 180 and 270) for every mode that it enumerates for the integrated panel.
- The rotation is required to be supported even if the integrated panel is in a duplicate or extended topology with another display device. For duplicate mode, it is acceptable to rotate all targets connected to the rotated source. Per path rotation is allowed but not required.

Both the above mentioned requirements are optional for Stereo 3D capable resolutions.

Send comments about this topic to Microsoft

System.Client.SystemConfiguration

In this topic:

System.Client.SystemConfiguration.Windows10RequiredComponents

Windows 10 systems must include certain devices.

Applies to	Windows 10 x64

	Windows 10 x86

Description

For all other Windows 10 systems, the table below lists the minimum required components to be present in a system in order for it to be compatible for Windows 10. All components must meet the compatibility requirements and pass device compatibility testing for Windows 10.

	Storage type	Meet minimum Microsoft Windows Storage requirements
System firmware		UEFI as defined in System.Fundamentals.Firmware requirements
Networking	Ethernet or Wi-Fi	Must be either a certified Ethernet or Wi-Fi adapter
Graphics	GPU	Minimum of Direct3D 10 Feature Level 9_3 and see System.Fundamentals.Graphics.WDDM

Send comments about this topic to Microsoft

System.Client.SystemImage

The requirements in this section describe the level two quality of HW + SW + OEM image

In this topic:

System.Client.SystemImage.SystemRecoveryEnvironment

System includes Windows Recovery Environment on a separate partition.

Applies to	Windows 10 x64 Windows 10 x86

Description

A system must include a separate partition with a bootable Windows Recovery Environment image file (winre.wim). The GPT partition should be of type PARTITION_MSFT_RECOVERY_GUID, includes the GPT_ATTRIBUTE_PLATFORM_REQUIRED and GPT_BASIC_DATA_ATTRIBUTE_NO_DRIVE_LETTER attributes, and contains at least 50 megabytes (MB) of free space after the Windows Recovery Environment image file has been copied to it.

Send comments about this topic to Microsoft

System.Client.SystemPartition

The requirements in this section describe the PC system partition configuration requirements.

In this topic:

System.Client.SystemPartition.DiskPartitioning

Systems that ship with a Windows operating system must meet partitioning requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

Windows systems must ship with an active system partition in addition to the operation system partition (configured as Boot, Page File, Crash Dump, etc.). This active system partition must have at least 250 MB of free space, above and beyond any space used by required files. This additional system partition can be used to host Windows Recovery Environment (RE) and OEM tools (provided by the OEM), so long as the partition still meets the 250 MB free space requirement.

Implementation of this partition allows support of current and future Windows features such as BitLocker, and simplifies configuration and deployments.

Tools and documentation to implement split-loader configuration can be found in **Windows OEM Preinstallation Kit/Automated Installation Kit (OPK/AIK).**

System.Client.SystemPartition.OEMPartition

Windows systems with recovery & OEM partitions must meet partitioning requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

If a system includes a separate partition for recovery purposes or an OEM partition for any other purpose, this separate partition must be identified with the GPT_ATTRIBUTE_PLATFORM_REQUIRED attribute. This attribute is defined as part of the PARTITION_INFORMATION_GPT (http://msdn.microsoft.com/en-us/library/aa365449(VS.85).aspx) structure.

For example:

- If this separate partition includes a bootable Windows Recovery Environment image file, the GPT partition must be of type PARTITION_MSFT_RECOVERY_GUID and include the GPT_ATTRIBUTE_PLATFORM_REQUIRED attribute.

Partitions which are identified with the GPT_ATTRIBUTE_PLATFORM_REQUIRED attribute must not be used for storing user data (such as through data backup, for example).

Send comments about this topic to Microsoft

System.Client.Tablet.Graphics

In this topic:

System.Client.Tablet.Graphics.SupportAllModeOrientations

Graphics drivers on Tablet systems are required to support all mode orientations.

Applies to	Windows 10 x64 Windows 10 x86

Description

Graphics drivers on tablet systems are required to support all mode orientations for every resolution enumerated for the integrated panel:

- A graphics driver is required to enumerate source modes for the integrated display. For each source mode enumerated the graphics driver is required to support each orientation (0, 90, 180 and 270).
- Each orientation is required even if the integrated panel is in a duplicate or extended topology with another display device. For duplicate mode, it is acceptable to rotate all targets connected to the rotated source. Per path rotation is allowed but not required.

Both the above mentioned requirements are optional for Stereo 3D capable resolutions.

Send comments about this topic to Microsoft

System.Client.WLAN.BasicConnectivity

In this topic:

System.Client.WLAN.BasicConnectivity.WlanBasicConnectivity

Applies to	Windows 10 x64 Windows 10 x86

Description:

If present WLAN allows for untethered connectivity to networks allowing for a wide range of scenarios such as browsing the web or streaming video content. When present the WLAN device must support, at a minimum, connecting to a WPA2 psk AES ap and all associated actions to enable making that connection. This includes the following:

- Scanning for available networks
- Device Enumeration
- Capabilities check
- Radio on / off
- Querying interface properties
- Connecting to a WPA2 PSK AES ap in the specified time as stated in the Windows 10 WLAN Requirements

Timing for the above actions can be found in the Windows 10 WLAN Device requirements.

Send comments about this topic to Microsoft

System.Client.WLAN.HangDetectionAndRecovery

In this topic:

- System.Client.WLAN.HangDetectionAndRecovery.WlanHangDetectionAndRecovery (If-Implemented) (WDI Drivers Only)

System.Client.WLAN.HangDetectionAndRecovery.WlanHangDetectionAndRecovery (If-Implemented) (WDI Drivers Only)

Applies to	Windows 10 x64 Windows 10 x86

Description:

Wi-Fi device firmware has been known to hang and / or get in a stuck state. Once that happens, the Lower Edge driver would either crash causing a 9F (Blue Screen) or the Wi-Fi subsystem gets into a state which requires a system reboot for the device to be functional again. In either case, the user is faced with a negative experience in their connectivity and their general system usage is disrupted. As an integral part of WDI, we have designed a mechanism to detect when the firmware gets into these states and recover the device seamlessly. This will ensure that user will see a minimal disruption in service by ensuring that the Wi-FI device stack recovers and resumes connectivity to the network without the system needing a reboot. Devices must report support for Hang Detection and Recovery in WDI_GET_ADAPTER_CAPABILITIES. Please refer to the WDI Spec for implementation details.

Requirement – Hardware / Firmware

System ACPI firmware: The System will provide the ACPI methods to PDLR the device either at a bus or at the device level.

System hardware: The system will allow for a PDLR (full device level reset). All systems must support PDLR.

System: The System will indicate support for PDLR support.

Device: The Lower Edge driver will be able to gather dumps with 25 ms and 250 Kb size

System: The system must complete the reset within 10 seconds.

Send comments about this topic to Microsoft

System.Client.WLAN.HostedNetwork

In this topic:

- System.Client.WLAN.HostedNetwork.WlanHostedNetwork (If-Implemented)

System.Client.WLAN.HostedNetwork.WlanHostedNetwork (If-Implemented)

Applies to	Windows 10 x64 Windows 10 x86

Description: With this feature, a Windows computer can use a single physical wireless adapter to connect as a client to a hardware access point (AP), while at the same time acting as a software AP allowing other wireless-capable devices to connect to it.

Send comments about this topic to Microsoft

System.Client.WLAN.Miracast

In this topic:

- System.Client.WLAN.Miracast.WlanMiracast (If-Implemented)

System.Client.WLAN.Miracast.WlanMiracast (If-Implemented)

Applies to	Windows 10 x64 Windows 10 x86

Description:

Miracast requires both WiFiDirect support in the WLAN Adapter and support in the Graphics driver. Miracast allows the user to extend their display to a Miracast supported sync device.

Send comments about this topic to Microsoft

System.Client.WLAN.WiFiDirect

In this topic:

- System.Client.WLAN.WiFiDirect.WlanWiFiDirect (If-Implemented)

System.Client.WLAN.WiFiDirect.WlanWiFiDirect (If-Implemented)

Applies to	Windows 10 x64 Windows 10 x86

Description:

Support for Wi-Fi Direct by the Wi-Fi Driver to enable Miracast, Public APIs for Wi-Fi Direct to allow pairing to & from the PC, Accepting and Connecting to other Wi-Fi Direct Device for the GO & the Client Role. This includes support for concurrent operation over Wi-Fi Direct & Station.

Send comments about this topic to Microsoft

System.Fundamentals.DebugPort

The ability to debug a system is crucial to supporting customers in the field and root-causing behavior in the kernel. Requirements in this area support the ability to kernel debug a Windows system.

In this topic:

System.Fundamentals.DebugPort.SystemExposesDebugInterface

System exposes debug interface that complies with Debug Port Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Windows 10 supports serveral different debug transports. They are listed below in the preferred order of implementation.

Hardware Debugging Transports

- Ethernet Network Interface Card from the supported list: http://msdn.microsoft.com/en-us/library/windows/hardware/hh830880
- USB 3.0 - xHCI controller compliant to xHCI debug specification.
- 1394 OHCI compliant Firewire controllers.
- USB2 OTG (on supported hardware for Windows, recommend XHCI debug instead).
- USB 2.0 EHCI debug (the debug enabled port must be user accessible).
- Legacy Serial (16550 compatible programming interface).

ADDITIONAL REQUIREMENTS

FOR ALL OF THE ABOVE IMPLEMENTATIONS THE FOLLOWING MUST APPLY:

- There must be at least one user accessible debug port on the machine. It is acceptable on systems which choose to not expose a USB port or any other acceptable port from the list above to instead require a separate debugging board or device that provides the ability to debug via one (or more) of the transports above. That device/board must terminate in the same standard port as would be used for the transport if it were 'onboard' the machine. If this device is required it must be documented in the system specifications, be user serviceable, be user installable on the machine, and available for sale from the machine's vendor.
- On retail PC platforms, it is strongly recommended that machines have 2 user accessible debug ports from the above list. The secondary debug port is required to debug scenarios where the first debug port is in use as part of the scenario. Microsoft is not responsible for debugging or servicing issues which cannot be debugged on the retail platform, or reproduced on development platforms.
- SoC development or prototype platforms provided to Microsoft for evaluation must have a dedicated debug port available for debugging. If the debug port is used for any scenarios that are expected to also be used on retail shipping devices, in that case, there must be a secondary debug port available for debugging. This is to ensure that SoC development platforms can be used to test and debug all scenarios for all available transports, including USB host and function.
- All debug device registers must be memory or I/O mapped. For example, the debug device must not be connected behind a shared bus such as SPI or I2C. This would prevent other devices on the same bus from being debugged.
- When enabled, the debug device shall be powered and clocked by the UEFI firmware during preboot, before transferring control to the boot block.

For additional information, see http://go.microsoft.com/fwlink/?LinkId=237141

Send comments about this topic to Microsoft

System.Fundamentals.DebugPort.USB

The ability to debug a USB3 system is crucial to supporting customers in the field and root-causing behavior in the kernel. Requirements in this area support the debugging capability for the xHCI controller based systems via a debug registers. Every system that has xHCI controller and USB3 external port should support via this port.

In this topic:

System.Fundamentals.DebugPort.USB.SystemExposesDebugInterfaceUsb

USB 3 system exposes debug interface that complies with Debug Port Specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems that support USB 3 are required to have xHCI controller(s) compliant to the xHCI debug specification. The xHCI controller(s) shall be memory mapped.

There must be at least one user accessible USB 3.0 debug port on the machine.

All USB 3.0 ports must protect against a short on the VBus pin such that if another USB host is connected, the USB circuitry is not damaged.

USB 3.0 hubs must not be integrated into the SoC or PCH/Southbridge.

For additional information, see http://go.microsoft.com/fwlink/?LinkId=58376.

Send comments about this topic to Microsoft

System.Fundamentals.EnergyEstimation

In this topic:

System.Fundamentals.EnergyEstimation.Discretional

There are currently three energy micro-benchmark tests in the HLK including primary storage, network, and primary display. These benchmarks are targeted to execute on any battery powered device. While in execution, the benchmarks emulate a set of steady state workloads of a particular component. At the same time, they also observe the battery drain.

- The battery must be nearly fully charged before executing a benchmark. A benchmark usually has multiple assessments. Before an assessment starts, the benchmark will estimate the expected runtime. If the remaining battery life duration is less than the time estimate required for executing the entire assessment, then the execution will immediately stop with an error message.
- Also note that some devices don't have a battery drain report correctly when it is at 100% capacity. Make sure it is less than or equal to 99% battery capacity remaining before executing a benchmark.
- Display benchmark requirements:

  Display benchmark tests the battery drain during different brightness settings. Therefore the device has to be able to adjust the brightness level through software control.

  - Open a command window as administrator
  - Run the following commands to see if the brightness changes:

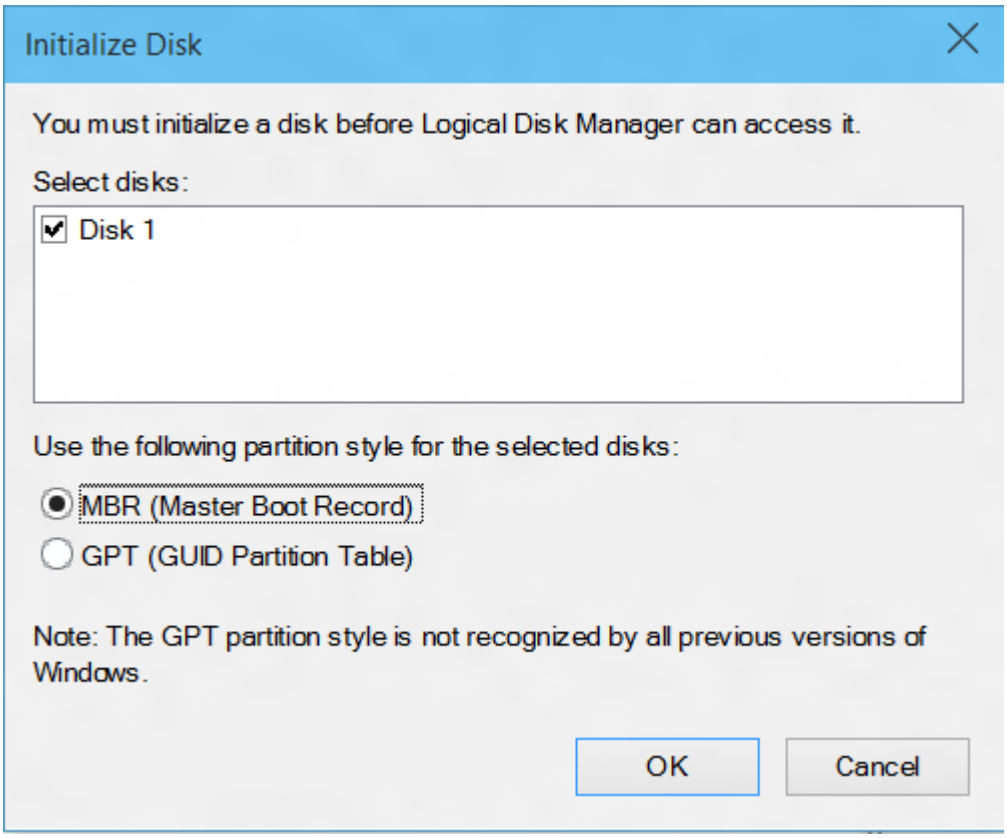
```
powercfg /setacvalueindex scheme_current sub_video aded5e82-b909-4619-9949-f5d71dac0bcb 10
powercfg /setdcvalueindex scheme_current sub_video aded5e82-b909-4619-9949-f5d71dac0bcb 10
powercfg /setactive scheme_current
```

  - If the brightness doesn't change, then the device is not suitable for this benchmark test.
- Network benchmark requirements: None
- Storage benchmark requirements:

  Storage benchmark needs to setup a fake drive get the baseline power.

  - This step needs the system with testsigning on and WTT service enabled. Once the test machine is set up through HLK controller, these should be automatically set up.
  - Open a command window as administrator, and cd to the folder of e3hlk\storhba. The storhba folder can also be found in <controller-name>\Tests\<processor architecture>\e3hlk.
  - cscript Scripts\Install_Storhba.wsf /storhba:1 /TestParameter:6 /LogicalUnitDiskSizeInMB:4096 /PhysicalLuns:1
  - Diskmgmt.msc (to open disk management).
  - Find the new disk with 4GB size.

- Initialize the disk using MBR partition style.



- Right click on it, select “New Simple Volume”. Follow the GUI to create a volume with 4GB space and format it in NTFS.
- Remember the assigned drive letter of the new disk, and close disk manager.
- Set\storapp_set.exe /flag –reset
- Once finish the test, you can remove the fake drive by running the following command:

```
cscript Scripts\Install_Storhba.wsf /storhba:0
```

There could be some problem of removing the fake driver after reset it. To work around it, you can create another fake drive first by running the following command:

```
Scripts\Install_Storhba.wsf /storhba:1 /TestParameter:6
/LogicalUnitDiskSizeInMB:4096 /PhysicalLuns:1
```

and then by running the following command:

```
Scripts\Install_Storhba.wsf /storhba:0
```

Send comments about this topic to Microsoft

System.Fundamentals.Firmware

In this topic:

System.Fundamentals.Firmware.ACPI

ACPI System Requirements

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All systems must meet the following ACPI table requirements.

ACPI Table Requirements	
Root System Description Pointer (RSDP)	Required

Root or Extended System Description Table (RSDT or XSDT)	Required
Fixed ACPI Description Table (FADT)	Revision 5 is required for Hardware-reduced ACPI platforms and systems that support connected standby platforms
Multiple APIC Description Table (MADT)	Required
Core System Resources Description (CSRT)	Required for ARM systems if non-Standard timers or any shared DMA controllers are exposed to the OS
Debug Port Table (DBGP)	Required. DBG2 table is required instead for Hardware-reduced ACPI platforms and systems that support connected standby platforms.
Differentiated System Description Table (DSDT)	Required

DSDT Requirements

As per ACPI 4.0a, all devices in the ACPI namespace must include:

- A vendor-assigned, ACPI-compliant Hardware ID (_HID object).
- A set of resources consumed (_CRS object).

In addition, the following conditional requirements apply:

- If any devices in the namespace share the same Hardware ID, then each is required to have a distinct Unique Identifier (_UID object).
- If any device in the namespace is enumerated by its parent bus (Plug and Play buses), the address of the device on its parent bus (_ADR object) is required.
- If any device in the namespace is compatible with a Microsoft-provided driver, the Compatible ID (_CID object) defined for that device type is required.

General-Purpose Input/Output (GPIO) on an System that Supports Connected Standby

GPIO Controllers for pins used by Windows drivers or ASL control methods must appear as devices in the ACPI namespace.

Devices in the namespace that are connected to GPIO pins on an enumerated controller device must:

- Include GPIO IO Connection resource descriptors in their _CRS for any GPIO I/O pins connected. Include GPIO Interrupt Connection resource descriptors in their _CRS object for any GPIO interrupt pins connected.

Simple Peripheral Bus (SPB) on an System that supports Connected Standby

SPB Controllers for connections used by Windows drivers or ASL control methods must appear as devices in the ACPI namespace.

Devices in the namespace that are connected to an enumerated SPB controller device (UART, I2C, SPI) must include SPB Connection resource descriptors in their _CRS for the SPB Connection(s) used.

Power Button

The power button, whether implemented as an ACPI Control Method Power Button or as part of the Windows-compatible Button Array, must:

- Be able to cause the system to power-up when required.
- Generate the Power Button Override Event (Section 4.7.2.2.1.3 of the ACPI 4.0a specification) when held down for 4 seconds.

Control Method Power Button

Systems dependent on built-in (or connected) keyboards/mice for input must conform to the ACPI Control Method Power Button (Section 4.7.2.2.1.2 of the ACPI 4.0a Specification). In addition, systems that support connected standby must:

- Implement the ACPI Control Method Power Button (Section 4.7.2.2.1.2 of the ACPI 4.0a Specification) using a dedicated GPIO interrupt pin to signal button press events.
- Configure the power button's GPIO interrupt pin as a non-shared, wake-capable (ExclusiveAndWake) GPIO interrupt connection resource.
- List the Power Button's GPIO interrupt connection resource in the ACPI Event Information (_AEI object) of the GPIO controller device to which it is connected.
- Provide the event method (_Lxx or _Exx object) for the power button event under the GPIO controller device in the ACPI namespace.

NOTE: For systems that require a separate driver to handle power button presses, it is acceptable to have that driver evaluate a control method that performs a Notify() on the Control Method Power Button device instead of using the GPIO-based solution above.

Button Array-based Power Button

Touch-first (keyboard-less) systems must:

- Implement the Windows-compatible Button Array device.
- Connect the power button to a dedicated GPIO interrupt pin.
- Configure the power button's GPIO interrupt pin as a non-shared, wake-capable (ExclusiveAndWake), Edge-triggered (Edge) GPIO interrupt connection resource, capable of interrupting on both edges (ActiveBoth).
- List the power button's GPIO Interrupt connection resource first in the Button Array device's _CRS object.

NOTE: For systems that require a separate driver to handle power button presses, it is acceptable to have that driver call the 5-Button array driver's power button event interface instead of using the GPIO-based solution above.

Time and Alarm Device

All battery-powered systems which are not capable of supporting Connected Standby are required to implement the Alarm capabilities of the ACPI Time and Alarm control method device.

Any system that supports Connected Standby that sets the "CMOS RTC Not Present" bit in the IAPC_BOOT_ARCH flags field of the FADT must implement the device's Time capabilities.

System.Fundamentals.Firmware.FirmwareSupportsBootingFromDVDDevice

System firmware supports booting from DVD device as defined by the El Torito specification

Target Feature System.Fundamentals.Firmware

Applies to	Windows Server 2016 Technical Preview x64

Description

The system firmware must support booting the system DVD if the system includes a DVD. The system firmware or option ROM must support the No-Emulation mode in the "El Torito" Bootable CD-ROM Format Specification, Version 1.0, for installing Windows® from optical media, such as bootable DVD media. The primary optical device must be bootable. This requirement applies to the primary optical storage and the primary bus to which the device is attached.

System.Fundamentals.Firmware.FirmwareSupportsUSBDevices

System firmware provides USB boot support for USB keyboards, mouse, and hubs

Applies to	Windows Server 2016 Technical Preview x64

Description

If the system includes support for USB keyboards and pointing devices, then the system firmware must: Support USB keyboards and pointing devices during system boot, resume from hibernate, and operating system setup and installation.

Support USB input devices at least two levels of physical hubs below the host controller.

Support composite input devices by the boot protocol as defined in HID.

For Windows Server systems, it is acceptable to enumerate, but not initialize all devices. If the device is accessed, it must be fully initialize before proceeding.

The USB controller and USB devices must be fully enumerated when:

- Anything other than the Windows Boot Manager is at the top of the system boot order
- A boot next variable has been set to boot to something other than the Windows Boot Manager
- On a system where the Windows Boot Manager is at the top of the list, an error case has been hit, such that the firmware fails over from the Windows Boot Manager to the next item in the list
- Resuming from hibernate, if the system was hibernated when booted from USB
- Firmware Setup is accessed.

System.Fundamentals.Firmware.HardwareMemoryReservation

System.Fundamentals.Firmware.HardwareMemoryReservation

Applies to	Windows 10 x64 Windows 10 x86

Description

This requirement limits the amount of memory that is reserved by the hardware (including drivers or firmware) and not available to the OS or user applications on a system. Taking into consideration the changes required to meet this requirement, it will be introduced in a phased manner. <=2GB systems – max of 3% + 25MB of 2GB (86.5MB)

- 2-3GB systems – max of 3% of 3GB + 25MB (117.2MB)
- For all other systems, a max of 120MB

- If screen resolution exceeds 1366x768, an additional 8 bytes per pixel will be allowed. Note: The budgets above are intended to cover 2 full screen video memory reservations for graphics drivers at 1366x768 at 32 bytes per pixel – 8MB. The adjustment above takes into consideration machines with higher resolutions.

RAM Size	Screen resolution	Threshold
1GB	1366x768	86.5MB
2GB	1366x768	86.5MB
2GB	1920x1080	86.5MB + 8MB
3GB	1366x768	117.2MB
3GB	1920x1080	117.2MB + 8MB
4GB	Any	120MB

Applies to Windows Client OS SKUs only

Design Notes:

- Hardware memory reservation is computed as the difference between the physical memory that is mapped as visible to the Windows OS (excluding all device/firmware reservations) compared to the installed RAM on the machine.

Installed memory is queried via Query GetPhysicallyInstalledSystemMemory() and OS visible memory is queried via GlobalMemoryStatusEx() – ullTotalPhys.

System.Fundamentals.Firmware.NoExternalDMAOnBoot

All external DMA ports must be off by default until the OS explicitly powers them through related controller(s).

Applies to	Windows 10 x64 Windows 10 x86

Description

The firmware must protect physical memory from unauthorized internal DMA (e.g. GPU accessing memory outside of video-specific memory) and all unauthorized DMA-capable external ports prior to

boot, during boot, and until such time as the OS powers up DMA ports via related bridge controllers. When the device enters a low-power state, DMA port device context must be saved, and restored upon returning to active state.

If the firmware has an option to enable and disable this protection, the shipping configuration must be with protection enabled, and turning protection off must be protected, for example with platform authentication via BIOS password.

Note that this requirement precludes the use of attached storage as boot media if it can only be accessed via an external DMA-capable port.

System.Fundamentals.Firmware.UEFIBitLocker

A system with TPM that supports wired LAN in pre-OS must support the UEFI 2.3.1 EFI_DHCP4_PROTOCOL protocol and the UEFI 2.3.1 EFI_DHCP6_PROTOCOL (and the corresponding service binding protocols).

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems which support TPM and wired LAN networking must support EFI_DHCP4_protocol, EFI_DHCP4_SERVICE_BINDING_PROTOCOL, EFI_DHCP6_protocol, and EFI_DHCP6_SERVICE_BINDING_PROTOCOL for wired LAN as defined in UEFI 2.3.1.

At pre-boot, BitLocker must be able to discover its Network Unlock provider on a Windows Deployment Server (WDS) via DHCP, and unlock the OS volume after retrieving a secret from WDS.

Details

All UEFI systems with TPM present and a wired LAN port must support BitLocker Network Unlock . This requires full DHCP support for wired LAN during preboot through a UEFI DHCP driver. Specifically, there must be UEFI driver implementations for EFI_DHCP4_protocol, EFI_DHCP4_SERVICE_BINDING_PROTOCOL , EFI_DHCP6_protocol, and EFI_DHCP6_SERVICE_BINDING_PROTOCOL for wired LAN, as defined in UEFI 2.3.1.

This requirement is "If Implemented" for Server systems and applies only if a Server system is UEFI capable

System.Fundamentals.Firmware.UEFIBootEntries

UEFI firmware honors software control over load option variables.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

UEFI systems must allow the Operating System to create both generic and device specific boot entries with Messaging Device path, specifically USB Class Device Path (UEFI version 2.3 main specification section 9.3.5.9). The firmware must respect these settings and not modify them once the OS has changed them. Furthermore, the firmware must accurately report the boot entries to the OS.

Functional Notes:

If the device corresponding to a boot entry is not found, it is preferable for the system to proceed to the next boot entry silently (without presenting an error message or requiring user intervention).

If the system is booted from an internal USB device and there is a USB class entry at the top of the boot order, the system should first attempt to boot from external USB devices before attempting internal USB boot devices.

Design Notes:

The UEFI specification requires that the software bootmanager be allowed to do the boot order programming (UEFI v. 2.3 Section 3.1.1 "Boot Manager Programming").

The firmware should interpret load options and device paths as specified in Section 9 "Protocols - Device Path Protocol."

The UEFI specification describes the variables that must be modifiable at runtime in Section 3.2, table 10.

The UEFI specification is available at http://www.UEFI.org.

This requirement is "If Implemented" for Server systems and applies only if a Server system is UEFI capable.

System.Fundamentals.Firmware.UEFICompatibility

System firmware must meet Windows Compatibility requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All systems which ship with a UEFI-compatible OS must be compatible with the following sections of the UEFI 2.3.1 specification:

2.3, 3.1, 4.3, 6.1 ~ 6.5, 7.1~7.5, 8.1, 8.2, 9.1, 9.5, 11.2 ~ 11.4, 11.8, 11.9, 12.4, 12.7, 12.8, 12.9, 18.5, 21.1, 21.3, 21.5, 27.1~27.8.

Additional guidance listed in "UEFI Support and Requirements: Microsoft Windows Server 2008" document (available at http://www.microsoft.com/whdc/system/platform/firmware/uefireg.mspx), if any, shall also be required.

All Windows 8 systems must boot in UEFI mode by default. Other requirements may add additional sections of compatibility to this list, but this is the baseline.

All systems, except servers, must be certified in UEFI mode without activating CSM. If a system is available with 32bit and/or 64bit UEFI, both configurations must be tested for certification. For server, certification in UEFI mode is only required if UEFI is implemented.

OEMs may ship with CSM mode activated and the enterprise or government customer's licensed OS selection when requested.

System.Fundamentals.Firmware.UEFIDefaultBoot

All client systems must be able to boot into UEFI boot mode and attempt to boot into this mode by default.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The System firmware must be able to achieve UEFI mode boot by default. Such a system may also support fallback to legacy BIOS mode boot for deploying OS images which do not support UEFI, if the user explicitly selects that option in the pre-boot UEFI BIOS menu.

This requirement is "If Implemented" for Server systems.

System.Fundamentals.Firmware.UEFILegacyFallback

System firmware must not fall back to legacy BIOS mode without explicit user action.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the system ships with a UEFI-compatible OS, system firmware must be implemented as UEFI and it must be able to achieve UEFI boot mode by default. Such a system may also support fallback to legacy BIOS boot on systems with OS which do not support UEFI, but only if the user selects that option in a pre-boot firmware user interface. Legacy option ROMs also may not be loaded by default.

"Explicit User Action" means that end user (or in case of enterprise customer, the IT pro) must manually access the pre-boot firmware configuration screen and change the setting. It may not ship in the BIOS mode by default and programmatic methods which can be attacked by malware are not acceptable.

All systems with Class 2 UEFI must not fall back to legacy BIOS mode nor load legacy Option ROM's without explicit user action within the pre-boot UEFI configuration UI."

An OEM may not ship a 64 bit system which defaults to legacy BIOS or loads legacy option ROMs if that system ships with a UEFI-compatible OS.

When Secure Boot is Enabled, Compatibility Support Modules (CSM) must NOT be loaded. Compatibility Support Modules are always prohibited on systems that support connected standby.

This requirement is "If Implemented" for Server systems and applies only if a Server system is UEFI capable.

System.Fundamentals.Firmware.UEFISecureBoot

All client systems must support UEFI Secure boot.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Note: These requirements are "If Implemented" for Server systems and apply only if a Server system supports UEFI Secure Boot.

- For the purposes of UEFI Secure Boot, the platform shall expose an interface to Secure Boot, whereby the system firmware is compliant with the following sections and sub-sections of UEFI version 2.3.1 Errata C:
  - 7.1
  - 7.2
  - 7.2.1
  - 27.2
  - 27.5 - 27.8 (as further profiled below).
- Secure Boot must ship enabled Configure UEFI Version 2.3.1 Errata C variables SecureBoot=1 and SetupMode=0 with a signature database (EFI_IMAGE_SECURITY_DATABASE) necessary to boot the machine securely pre-provisioned, and include a PK that is set and a valid KEK database. The system uses this database to verify that only trusted code (for example: trusted signed boot loader) is initialized, and that any unsigned image or an image that is signed by an unauthorized publisher does not execute. The contents of the signature database is determined by the OEM, based on the required native and third- party UEFI drivers, respective recovery needs, and the OS Boot Loader installed on the machine. The following Microsoft-provided EFI_CERT_X509 signature shall be included in the signature database: "CN=Microsoft Windows Production PCA 2011" and "Cert Hash(sha1): 58 0a 6f 4c c4 e4 b6 69 b9 eb dc 1b 2b 3e 08 7b 80 d0 67 8d" which shall use the following

SignatureOwner GUID: {77fa9abd-0359-4d32- bd60-28f4e78f784b}, must also be included in the form of either an EFI_CERT_X509_GUID or EFI_CERT_RSA2048_GUID type:

  - Note: Must NOT contain the following certificate: "CN=Microsoft Windows PCA 2010" and "Cert Hash(sha1): c0 13 86 a9 07 49 64 04 f2 76 c3 c1 85 3a bf 4a 52 74 af 88"
  - Note II: Windows Server systems may ship with Secure Boot disabled, but all other provisions of this sub-requirement must be met

- When Secure Boot is Enabled, Compatibility Support Modules (CSM) must NOT be loaded.
- The initial UEFI signature databases (db) shall be created with the EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS attribute stored in firmware flash and may be updated only with an OEM-signed firmware update or through UEFI authenticated variable write.
- Support for the UEFI "forbidden" signature database (EFI_IMAGE_SECURITY_DATABASE1) must be implemented.
- The platform shall ship with a "forbidden" signature database (EFI_IMAGE_SECURITY_DATABASE1) created with the EFI_VARIABLE_TIME_BASED_AUTHENTICATED_ACCESS attribute. When a signature is added to the forbidden signature database, upon reboot, any image certified with that signature must not be allowed to initialize/execute.
- Secure Boot must be rooted in a protected or ROM-based Public Key. Secure Boot must be rooted in an RSA public key with a modulus size of at least 2048 bits, and either be based in unalterable ROM or otherwise protected from alteration by a secure firmware update process, as defined below.
- Secure firmware update process. If the platform firmware is to be serviced, it must follow a secure update process. To ensure the lowest level code layer is not compromised, the platform must support a secure firmware update process that ensures only signed firmware components that can be verified using the signature database (and are not invalidated by the forbidden signature database) can be installed. UEFI Boot Services variables must be hardware-protected and preserved across flash updates. The Flash ROM that stores the UEFI BIOS code must be protected. Flash that is typically open at reset (to allow for authenticated firmware updates) must subsequently be locked before running any unauthorized code. The firmware update process must also protect against rolling back to insecure versions, or non-production versions that may disable secure boot or include non-production keys. A

physically present user may however override the rollback protection manually. In such a scenario (where the rollback protection is overridden), the TPM must be cleared. Further, it is recommended that manufacturers writing BIOS code adhere to the NIST guidelines set out in NIST SP 800-147 (http://csrc.nist.gov/publications/nistpubs/800-147/NIST-SP800-147-April2011.pdf), BIOS Protection Guidelines, which provides guidelines for building features into the BIOS that help protect it from being modified or corrupted by attackers. For example, by using cryptographic digital signatures to authenticate BIOS updates.

- Signed Firmware Code Integrity Check. Firmware that is installed by the OEM and is either read-only or protected by a secure firmware update process, as defined above, may be considered protected. Systems shall verify that all unprotected firmware components, UEFI drivers, and UEFI applications are signed using minimum RSA-2048 with SHA-256 (MD5 and SHA-1 are prohibited), and verify that UEFI applications and drivers that are not signed as per these requirements will fail to run (this is the default policy for acceptable signature algorithms). If an images signature is not found in the authorized database, or is found in the forbidden database, the image must not be started, and instead, information about it shall be placed in the Image Execution Information Table.

- UEFI firmware and driver implementations must be resistant to malicious input from untrusted sources. Incomplete input validation may result in buffer overflows, integer and pointer corruption, memory overwrites, and other vulnerabilities, compromising the runtime integrity of authenticated UEFI components.

- Verify Signature of all Boot Apps and Boot Loaders. Upon power-on, the platform shall start executing boot firmware and use public key cryptography as per algorithm policy to verify the signatures of all images in the boot sequence up-to and including the Windows Boot Manager.

- Microsoft Key Encryption Key (KEK) is provisioned A valid Microsoft-provided KEK is included in the KEK database. Microsoft provides the KEK in the form of either an EFI_CERT_X509_GUID or EFI_CERT_RSA2048_GUID type signature. The Microsoft KEK signature uses the following SignatureOwner GUID: {77fa9abd-0359-4d32-bd60-28f4e78f784b}.

- PKpub verification. The PKpub key is owned by the OEM and stored in firmware flash. The private-key counterpart to PKpub is PKpriv, which controls Secure Boot policy on all OEM-manufactured devices, and its protection and use must be secured against un-authorized use

or disclosure. PKpub must exist and the operating system must be able to read the value and verify that it exists with proper key length.

- No in-line mechanism is provided whereby a user can bypass Secure Boot failures and boot anyway Signature verification override during boot when Secure Boot is enabled is not allowed. A physically present user override is not permitted for UEFI images that fail signature verification during boot. If a user wants to boot an image that does not pass signature verification, they must explicitly disable Secure Boot on the target system.
- UEFI Shells and related applications. UEFI Modules that are not required to boot the platform must not be signed by any production certificate stored in "db", as UEFI applications can weaken the security of Secure Boot. For example, this includes and is not limited to UEFI Shells as well as manufacturing, test, debug, RMA, & decommissioning tools. Running these tools and shells must require that a platform administrator disables Secure Boot.
- Secure Boot Variable. The firmware shall implement the SecureBoot variable as documented in Section 3.2 "Globally Defined Variables' of UEFI Specification Version 2.3.1 Errata C"

  For devices which are designed to always boot with a specific Secure Boot configuration, the two requirements below to support Custom Mode and the ability to disable secure boot are optional.
- On non-ARM systems, the platform MUST implement the ability for a physically present user to select between two Secure Boot modes in firmware setup: "Custom" and "Standard". Custom Mode allows for more flexibility as specified in the following:
  - It shall be possible for a physically present user to use the Custom Mode firmware setup option to modify the contents of the Secure Boot signature databases and the PK. This may be implemented by simply providing the option to clear all Secure Boot databases (PK, KEK, db, dbx), which puts the system into setup mode.
  - If the user ends up deleting the PK then, upon exiting the Custom Mode firmware setup, the system is operating in Setup Mode with SecureBoot turned off.
  - The firmware setup shall indicate if Secure Boot is turned on, and if it is operated in Standard or Custom Mode. The firmware setup must provide an option to return from Custom to Standard Mode which restores the factory defaults. On an ARM system, it is forbidden to enable Custom Mode. Only Standard Mode may be enabled.

- Enable/Disable Secure Boot. A physically present user must be allowed to disable Secure Boot via firmware setup without possession of PKpriv. A Windows Server may also disable Secure Boot remotely using a strongly authenticated (preferably public-key based) out-of-band management connection, such as to a baseboard management controller or service processor. Programmatic disabling of Secure Boot either during Boot Services or after exiting EFI Boot Services MUST NOT be possible. Disabling Secure Boot must not be possible on ARM systems.
- If the firmware is reset to factory defaults, then any customized Secure Boot variables are also factory reset. If the firmware settings are reset to factory defaults, all custom-set variables shall be erased and the OEM PKpub shall be re-established along with the original, manufacturer-provisioned signature databases.
- OEM mechanism exists to remediate failed EFI boot components up to and including the Windows OS loader (bootmgr.efi). Images in the EFI boot path that fail Secure Boot signature verification MUST not be executed, and the EFI_IMAGE_EXECUTION_INFO_TABLE entry for that component shall be updated with the reason for the failure. The UEFI boot manager shall initiate recovery according to an OEM-specific strategy for all components up to and including Windows bootmgr.efi.
- A working Windows RE image must be present on all Windows 8 client systems The Windows Recovery image must be present in the factory image on every Secure Boot capable system. To support automated recovery and provide a positive user experience on Secure Boot systems, the Windows RE image must be present and enabled by default. As part of the Windows 8 Trusted Boot work enhancements have been made to Windows RE to allow optimized recovery from signature verification failures in Secure Boot. OEMs must include Windows RE as part of their factory image on all Windows 8 client systems.
- Firmware-based backup and restore. If the OEM provides a mechanism to backup boot critical files (for example: EFI drivers and boot applications), it must be in a secure location only accessible and serviceable by firmware. The OEM may provide the capacity via firmware or other backup store to store backup copies of boot critical files and recovery tools. If such a store is implemented, the solution must also have the capability to restore the target files onto the system without the need for external media or user intervention. This is a differentiator for the OEM in failover protection, used if the Windows OS loader (bootmgr.efi) or other boot critical components fail, preventing Windows native recovery solutions to execute.

- Firmware-based backup synchronization. Backup copies of boot critical components (for example: EFI drivers and boot applications) stored in firmware must be serviced in sync with updates to same files on the system If the system has the capability to store a backup copy of the Windows OS loader (bootmgr.efi), and potentially other critical boot components, then the files must be serviced on the same schedule as their counterparts in use on the live system. If the Windows OS loader is updated by Windows Update, then the backup copy of bootmgr.efi stored in firmware must be updated on the next boot.
- All Windows 8 client systems must support a secondary boot path. For all Windows 8 systems configured for Secure Boot, there must be an alternate boot path option that is followed by the firmware in the event that the primary Windows OS loader fails. The second boot path may point either to the default shadow copy installed by Windows to the system backup store (<EFI System Volume>\EFI\Boot\boot<platform>.efi), or to a copy stored by the OEM firmware-based mechanism. This alternate path could be a file in executable memory, or point to a firmware-based remediation process that rolls a copy out of the OEM predetermined backup store.
- All Windows 8 client systems must support a USB boot path for recovery purposes. For all Windows 8 systems configured for Secure Boot, there is a last resort of booting from USB.
- Supporting GetVariable() for the EFI_IMAGE_SECURITY_DATABASE (both authorized and forbidden signature database) and the SecureBoot variable.
- Supporting SetVariable() for the EFI_IMAGE_SECURITY_DATABASE (both authorized and forbidden signature database), using an authorized KEK for authentication.
- Reserved Memory for Windows Secure Boot UEFI Variables. A total of at least 64 KB of non-volatile NVRAM storage memory must be reserved for NV UEFI variables (authenticated and unauthenticated, BS and RT) used by UEFI Secure Boot and Windows, and the maximum supported variable size must be at least 32kB. There is no maximum NVRAM storage limit.
- During normal firmware updates the following must be preserved:
  - The Secure Boot state & configuration (PK, KEK, db, dbx, SetupMode, SecureBoot)
  - All UEFI variables with VendorGuid {77fa9abd-0359-4d32-bd60-28f4e78f784b}
  - A physically-present user who authenticates to the firmware may change, reset, or delete these values
- The platform shall support EFI variables that are:

- accessible only during the boot services or also accessible in the runtime phase;
- non-volatile; and
- Possible to update only after proper authorization, for example, being properly signed.

The platform must support EFI variables with any valid combination of the following UEFI 2.3.1 variable attributes set:

Copy

EFI_VARIABLE_NON_VOLATILE

EFI_VARIABLE_BOOTSERVICE_ACCESS

EFI_VARIABLE_RUNTIME_ACCESS

EFI_VARIABLE_AUTHENTICATED_WRITE_ACCESS

EFI_VARIABLE_APPEND_WRITE

EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS

Connected Standby systems must meet all of the requirements cited in both "system.fundamentals.firmware.uefisecureboot" and "system.fundamentals.firmware.uefisecureboot.connectedstandby".

The documents referenced in this document may be requested by contacting http://go.microsoft.com/fwlink/p/?LinkId=237130

- Microsoft UEFI CA key MUST be included in SecureBoot DB. Platform may choose to not include Microsoft UEFI CA key only if platform by design blocks all the 3rd party UEFI extensions.
- Optional: Platform shall expose dbDefault, dbxDefault, KEKDefault, & PKDefault to be accessible for read through OS for test purposes

System.Fundamentals.Firmware.UEFITimingClass

System firmware must expose timing and class information.

Applies to	Windows 10 x64 Windows 10 x86

Description

- During POST, the firmware shall measure its own timing and record the duration of post, rounded to the nearest mSec.
- These timings shall measure tEnd of reset sequence (Timer value noted at beginning of BIOS initialization - typically at reset vector) Handoff to OS Loader.

System.Fundamentals.Firmware.Update

System firmware must meet the requirements in order to support system and/or device firmware updates using firmware driver package.

Applies to	Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) x64 Windows 10 for desktop editions x86 Windows Server 2016 Technical Preview x64

Description

These requirements must be met by any system that updates system and/or device firmware using the Windows firmware driver package mechanism.

- The ESRT table must define at least one firmware resource (ESRE) in the resource list, which must include a system firmware resource.
- Only one system firmware resource can be defined in the ESRT.
- No two resources in the ESRT table are permitted to have the same firmware class GUID.
- ESRE must provide appropriate status code including success or failed firmware update attempt, on the subsequent boot, to the OS.
- Firmware for every resource defined by the ESRT must be upgradable to a newer version
- Firmware version of a particular resource must not break compatibility with firmware versions of other resources.
- Firmware must provide the lowest supported firmware version using the field "LowestSupportedFirmwareVersion" in the ESRE table. Firmware must not allow rollback to any version lower than the lowest supported version. Whenever a security related update has successfully been made, this field must be updated to match the "FirmwareVersion" field in the ESRE. When the lowest firmware version does not match the current firmware version, firmware must allow rollbacks to any version between the current version and the lowest supported version (inclusive).
- Firmware must seamlessly recover from failed update attempts if it is not able to transfer control to the OS after an update is applied.

Send comments about this topic to Microsoft

System.Fundamentals.Firmware.Boot

This section describes boot requirements for all client systems.

In this topic:

System.Fundamentals.Firmware.Boot.EitherGraphicsAdapter

System firmware must be able to boot a system with onboard or integrated graphics and with multiple graphics adapters.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems with GPUs on the system board and mobile systems that can use a docking station with PCI slots must provide a means in the system firmware setup utility to compel the system to use the onboard graphics device to boot. This capability is required so the onboard graphics device can be used in a multiple-monitor configuration and for hot undocking a mobile system.

If the system includes PCI, AGP, or PCI Express expansion slots, the system firmware must be able to boot a system with multiple graphics adapters. The system BIOS must designate one device as the VGA device and disable VGA on all other adapters. A system with an integrated graphics chipset and one or more discrete graphics adapters must be able to disable the integrated graphics chipset if the integrated graphics chipset cannot function as a non-VGA chipset.

System.Fundamentals.Firmware.Boot.SystemWithBootDeviceGreaterThan

Systems with a boot device with a capacity greater than 2.2 terabytes must comply with requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems with a boot device with a capacity greater than 2.2 terabytes must comply with the following requirements:

- The system must be 64-bit.

- The system must comply with the UEFI requirements in the section system.fundamentals.firmware.
- The system must comply with Advanced Configuration and Power Interface (ACPI) Specification version 4.0. Specifically, the system must be able to support legacy or Operating System-directed configuration and Power Management (OSPM)/ACPI mode.

Send comments about this topic to Microsoft

System.Fundamentals.Firmware.CS

In this topic:

System.Fundamentals.Firmware.CS.CryptoCapabilities

System that support Connected Standby must include cryptographic capabilities to meet customer expectations on platform speed and performance.

Applies to	Windows 10 x64 Windows 10 x86

Description

Since all components in the boot path as well as many performance-critical OS subsystems will invoke cryptographic functions, run-time performance of these functions is critical. The following requirements have been drafted to help ensure sufficient cryptographic capabilities are in place to meet customer expectations on platform speed and performance:

- The platform must meet cryptographic performance requirements as stated in Table 1. The platform may meet these requirements through any combination of hardware or software. The following general remarks apply to all algorithms in Table below:
  - Target performance must be achieved in a multi-threaded test. The number of threads will be determined by querying the property named L"HardwareThreads" on the BCrypt provider through the CNG BCryptGetProperty interface. The provider is required to return

a DWORD value in response. If the provider does not support this property, the test will run single-threaded.

- When cryptographic acceleration engines are used: Due to the overhead involved in dispatching requests to hardware acceleration engines, it is recommended that small requests be handled in software. Similarly, it is recommended that vendors consider using CPU-based cryptography to improve throughput when all cryptographic acceleration engines are fully utilized, idle capacity is available on the CPU, and the device is in a high-performance mode (such as when connected to AC power).

- ARM based platforms must implement the EFI_HASH_PROTOCOL from UEFI Industry Group, Unified Extensible Firmware Interface Specification version 2.3.1 Errata B. The EFI_HASH_PROTOCOL implementation must be accessible from Windows pre-Operating System code (i.e. in the Boot Services phase of platform boot). Both the UEFI hash protocols EFI_HASH_ALGORITHM_SHA1_NOPAD_GUID and EFI_HASH_ALGORITHM_SHA256_NOPAD_GUID must be supported, and the implementation must support passing a Message at least 10 Mbytes long (Note: No padding must be applied at any point to the input data).

- To make entropy generation capabilities available to Windows pre-Operating System code, the platform shall support the EFI_RNG_PROTOCOL for pre-Operating System read of at least 256 bits of entropy in a single call (i.e. 256 bits of full entropy from a source with security strength of at least 256 bits). The protocol definition can be found in Microsoft Corporation, "UEFI Entropy-Gathering Protocol,"2.

- All cryptographic capabilities in accordance with Table 1 shall be accessible from the runtime OS in kernel mode, through the interface specified in Microsoft Corporation, "BCrypt Profile for SoC Acceleration," 2 .

- OPTIONAL. It is recommended that the platform's cryptographic capabilities also be accessible from the runtime OS in user mode, through the interface previously referenced in Requirement 4.

- The OS interface library shall be implemented in such a way that when an unprivileged process is operating on a given key in a given context, it shall not be able to access the key material or perform key operations associated with other contexts.

- OPTIONAL. It is highly recommended that the RNG capability of the platform be exposed through an OS entropy source through the interface specified in Microsoft Corporation, "BCrypt Profile for SoC Acceleration," previously referenced in Requirement 4.
- OPTIONAL. (Applies when a cryptographic acceleration engine is used) It should be possible to maintain and perform cryptographic operations on at least three distinct symmetric keys or two symmetric keys and one asymmetric key simultaneously in the acceleration engine.

Table: Algorithm-specific requirements. The "Category" column classifies algorithms as mandatory to support at the software interface as per requirement 4 (M), or optional (O). Note that all algorithms that are accelerated in hardware must also be exposed through the software interface.

Algorithm	Category	Modes	Mandatory Supported Key Size(s)	Remarks
3-DES	O	ECB, CBC, CFB8	112, 168	
AES	M	ECB, CBC, CFB8, CFB128, CCM, CMAC, GCM, GMAC	128, 192, 256	Performance >= 60 MBytes/s for AES-128-CBC and AES-128-ECB encryption and decryption as measured at the CNG kernel-mode BCrypt interface when processing 32 kByte blocks.
	O	CTR, XTS, IAPM	128, 192, 256	
RSA	O	PKCS #1 v1.5, PSS, OAEP	512 to 16384 in 8-byte increments	Public key performance for 2048-bit keys (and public exponent F4 (0x10001)) when verifying PKCS#1v1.5 padded signatures, measured at the kernel mode BCrypt interface <=0.6 ms/verification.
ECC	O	ECDSA, ECDH	256	If implemented, must support Elliptic curve P-256 defined in National Institute for Standards and Technology, "Digital Signature Standard," FIPS 186-3, June 2009.
SHA-1	O			Performance >60 Mbytes/s as measured at the CNG kernel-mode BCrypt interface when processing 4kByte blocks.

HMAC-SHA1	O			
SHA-256	O			Performance >60 Mbytes/s as measured at the CNG kernel-mode BCrypt interface when processing 4kByte blocks.
HMAC SHA-256	O			
RNG	M	Entropy source with optional FIPS 800-90-based DRBG		Security strength must be at least 256 bits1. Note that exposing this functionality through the UEFI Entropy-Gathering Protocol is required (see Req 3) and exposing it as an OS entropy source is recommended (see Req 7).

1The Connected Standby vendor shall supply documentation indicating, and allowing Microsoft to estimate, the quality of the entropy source. The entropy shall be assessed using the min-entropy method of Appendix C of National Institute for Standards and Technology, "Recommendation for Random Number Generation using Deterministic Random Bit Generators," FIPS 800-90, March 2007 and must surpass or be equal to 256 bits before the runtime OS starts.

2This specification must be requested explicitly from Microsoft. To request the current version, please contact http://go.microsoft.com/fwlink/?LinkId=237130.

System.Fundamentals.Firmware.CS.UEFISecureBoot.ConnectedStandby

All client systems that support Connected Standby must support UEFI Secure Boot.

Applies to	Windows 10 x64 Windows 10 x86

Description

- Connected standby systems must meet all of the requirements cited in this section and under System.Fundamentals.Firmware.Uefisecureboot section. In addition MUST meet the following requirement.
- Boot Integrity. Platform uses on-die ROM or One-Time Programmable (OTP) memory for storing initial boot code and initial public key (or hash of initial public key) used to provide

boot integrity, and provides power-on reset logic to execute from on-die ROM or secure on-die SRAM.

- Secure Boot launch of Windows 8 BootMgr must not require use of an Allowed DB entry other than the Microsoft-provided EFI_CERT_X509 signature with "CN=Microsoft Windows Production PCA 2011" and "Cert Hash(sha1): 58 0a 6f 4c c4 e4 b6 69 b9 eb dc 1b 2b 3e 08 7b 80 d0 67 8d.
- On ARM, the UEFI Allowed database ("db") must not contain any other entry than the Microsoft- provided EFI_CERT_X509 signature with "CN=Microsoft Windows Production PCA 2011" and "Cert Hash(sha1): 58 0a 6f 4c c4 e4 b6 69 b9 eb dc 1b 2b 3e 08 7b 80 d0 67 8d.
- The policy for acceptable signature algorithms (and padding schemes) shall be possible to update. The exact method for updating the policy is determined by each authority (for example: Microsoft determines policies for binaries it is responsible for; SOC vendor for firmware updates). It is recognized that the initial ROM code need not have an ability to update the initial signature scheme.
- The platform shall maintain and enforce a policy with regards to signature authorities for firmware and pre-Operating System components; the policy (and hence the set of authorities) shall be possible to update. The update must happen either as a result of actions by a physically present authorized user or by providing a policy update signed by an existing authority authorized for this task. On ARM platforms, the physical presence alone is not sufficient. Signature authority (db or KEK) updates must be authenticated on ARM platforms.
- Upon power-on, the platform shall start executing read-only boot firmware stored on-die and use public key cryptography as per algorithm policy to verify the signatures of all images in the boot sequence up- to the Windows Boot Manager.
- Protection of physical memory from unauthorized internal DMA (for example: GPU accessing memory outside of video-specific memory) and all external DMA access to the SOC. The firmware shall enable this protection as early as feasible, preferably within the initial boot firmware.
- Optional: The memory containing the initial boot firmware (executing in SRAM) may be made inaccessible upon jumping to the next validated stage of the boot sequence. The initial boot firmware may remain inaccessible until power-on-reset is triggered.
- The platform shall enforce policy regarding the replacement of firmware components. The policy must include protection against rollback. It is left to the platform vendor to define the

exact method for policy enforcement, but the signature verification of all firmware updates must pass and the update must be identified in such a manner that a later version of a component cannot, without proper authorization (for example: physical presence), be replaced by an earlier version of the component where earlier and later may be defined by a (signed) version number, for example.

- Optional: The platform shall offer at least 112 logical eFuse bits to support platform firmware revision control in accordance with the above requirement.
- Physical Security Requirements. In retail parts, once the platform is configured for Production mode, the hardware must disable all external hardware debug interfaces such as JTAG that may be used to modify the platform's security state, and disable all hardware test modes and disable all scan chains. The disabling must be permanent unless re-enablement unconditionally causes all device-managed keys that impact secure boot, TPM, and storage security to be rendered permanently erased.
- On ARM platforms Secure Boot Custom Mode is not allowed.
- A physically present user cannot override Secure Boot authenticated variables (for example: PK, KEK, db, dbx).
- Platforms shall be UEFI Class three (see UEFI Industry Group, Evaluating UEFI using Commercially Available Platforms and Solutions, version 0.3, for a definition) with no Compatibility Support Module installed or installable. BIOS emulation and legacy PC/AT boot must be disabled.
- Each device is required to leave manufacturing provisioned with all cryptographic seeds and keys that are necessary to prevent attacks against the device’s Secure Boot, TPM and secure persistent storage implementations. Seeds and symmetric keys shall be immutable, per-device-unique, and non- predictable (random with sufficient length to resist exhaustive search; see NIST 800-31A for acceptable key sizes).
- The platform is required to implement hardware security test interface and share documentation and tools as specified in the ‘Hardware Security Test Interface Specification’ document. This requirement is IF IMPLEMENTED for Server.

Send comments about this topic to Microsoft

System.Fundamentals.Firmware.CS.UEFISecureBoot

Connected standby systems have additional UEFI Secure Boot requirements.

In this topic:

System.Fundamentals.Firmware.CS.UEFISecureBoot.Provisioning

Systems are required to leave manufacturing provisioned with all cryptographic seeds and keys that are necessary to prevent attacks

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Each device is required to leave manufacturing provisioned with all cryptographic seeds and keys that are necessary to prevent attacks against the device's Secure Boot, TPM and secure persistent storage implementations. Seeds and symmetric keys shall be immutable, per-device-unique, and non- predictable (random with sufficient length to resist exhaustive search; see NIST 800-31A for acceptable key sizes).

The platform is required to implement a hardware interface and share documentation and tools as specified in the 'Hardware Security Testability Specification' document (This document is available on Connect).

This requirement is IF IMPLEMENTED for Server.

Send comments about this topic to Microsoft

System.Fundamentals.Firmware.TPR

This feature includes requirements specific to system firmware with eDrive support.

In this topic:

System.Fundamentals.Firmware.TPR.UEFIEncryptedHDD

Systems which ship with a self-encrypting hard drive as a storage device must support the UEFI 2.3.1 EFI_STORAGE_SECURITY_COMMAND_PROTOCOL protocols and shall contain a non-OS partition that can be used to store WinRE.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If self-encrypted drive support is implemented it must have a UEFI-compatible OS and contain system firmware both conforming to system firmware logo requirements as defined in System.Fundamentals.Firmware and also contain a separate WINRE partition.

BitLocker shall support self-encrypting drivers that conform to the eDrive Device Guidelines available on WHDC at http://msdn.microsoft.com/en-us/library/windows/hardware/br259095

UEFI – Trusted Command Support in UEFI 2.3 + UEFI Mantis change number 616 or UEFI 2.3.1

Standard v2.3 + errata Bv2 www.uefi.org

Mantis Change Number 616 www.uefi.org (This is not part of v2.3)

All necessary partitions have to be created, managed individually pre/post encryption. The WINRE partition must always be separate and outside of the OS/encryption partition.

If WinRE is on the system partition, the size is 350 MB. If it's not the system partition, then it's 300MB. This is assuming MBR layout. (For GPT, WinRE is always separate from the ESP, therefore 300 MB.)

This requirement is "If Implemented" for Server systems and applies only if a Server system is UEFI capable.

Send comments about this topic to Microsoft

System.Fundamentals.Graphics

In this topic:

System.Fundamentals.Graphics.FirmwareSupportsLargeAperture

32-bit and 64-bit system firmware supports large aperture graphic adapters.

Applies to	Windows 10 x64

	Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The system firmware (BIOS/UEFI) must support large aperture graphics adapters. The 32-bit system firmware (BIOS/UEFI) must be able to support at least 256 MB aperture. On 64-bit systems the firmware (BIOS/UEFI) must be able to support at least 1GB aperture.

A system that supports multiple graphics adapters must ensure sufficient resources for each adapter. For example on a 32bit system with 4 graphics adapters, each adapter must receive at least 256 MB memory resources each on the PCI bus.

System.Fundamentals.Graphics.MicrosoftBasicDisplayDriver

System is compatible with the Microsoft Basic Display Driver.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The System must boot in a mode where the frame buffer used by the Microsoft basic display driver is displayed whenever the Microsoft display driver writes to the frame buffer. No other driver is involved to accomplish this output. The frame buffer must be linear and in BGRA format.

System.Fundamentals.Graphics.NoRebootUpgrade

Graphics drivers must be upgradable without a reboot of the system.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The WDDM driver model has supported rebootless upgrade since Windows Vista. For Windows all systems must support the upgrade of graphics driver package without requiring the system to reboot.

For example the graphics driver package includes the graphics driver and all associated utilities and services.

System.Fundamentals.Graphics.PremiumContentPlayback

Protected Environment Signature requirement.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems must be capable of premium (i.e., protected) audio or video content playback. If one or more (Audio Processing Objects) APOs are provided in the audio driver, the APOs must be able to load successfully without disabling the Audio Device Graph as a protected process.

Send comments about this topic to Microsoft

System.Fundamentals.Graphics.DisplayRender

The requirements in this section are enforced on any graphics device implementing display and render portion of the WDDM.

In this topic:

System.Fundamentals.Graphics.DisplayRender.StableAndFunctional

Display device functions properly and does not generate hangs or faults under prolonged stress.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The system must run under prolonged stress without generating hangs or faults.

Send comments about this topic to Microsoft

System.Fundamentals.Graphics.HybridGraphics

Hybrid Graphics Feature

In this topic:

System.Fundamentals.Graphics.HybridGraphics.MultiGPU

Hybrid Graphics

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

New systems shipping in Windows 10 that expect to be hybrid capable must adhere to the following requirements:

- A Microsoft supported hybrid system can only be composed as one integrated GPU (iGPU) and one discrete GPU (dGPU)
- Cannot have more than two GPUs
- Both GPU's must be DX11.0 or higher
- Both GPU's driver must be WDDM 2.0 or higher
- Both GPUs must implement the standard allocation type added to the KM and associated UM DDIs to support cross adapter shared surfaces.
- If each GPU has separate standard drivers, then they must be independent of each other and able to be updated independently without breaking hybrid functionality
- The dGPU must be equal or higher performance than the iGPU
- The dGPU adapter is the one that sets the discrete hybrid cap

All other multi-GPU configurations do not get Microsoft hybrid support. They will be treated the same way as defined by the "System.Fundamentals.Graphics.MultipleOperatingMode" requirement.

D-list requirements

This is the list of applications maintained by the dGPU IHV for choosing a GPU for an app to run in hybrid mode or not:

- The D-list DLL can be loaded into a process and queried at most once during D3D initialization
- The DLL size must be under 200KB

- The DLL must be able to return the GPU selection choice within 4ms

Power Management Requirements

Following are the power management requirements for the discrete GPU participating in a hybrid configuration:

- The driver is required to register for runtime power management.
- The driver needs to register certain power components based on the following scenarios.

Device does not require D3 transitions	DXGK_POWER_COMPONENT_ENGINE component for each GPU engine (node) A DXGK_POWER_COMPONENT_D3_TRANSITION component with one F state
Device requires D3 transitions and has no self-refresh memory	A DXGK_POWER_COMPONENT_D3_TRANSITION component with two F states DXGK_POWER_COMPONENT_ENGINE component for each GPU engine (node) A DXGK_POWER_COMPONENT_MEMORY component for each memory segment. If TransitionalLatency of this component is > 200us, component must also have DXGK_POWER_COMPONENT_FLAGS::DriverCompletesFStateTransition flag set
Device requires D3 transitions and has self-refresh memory	A DXGK_POWER_COMPONENT_D3_TRANSITION component with two F states DXGK_POWER_COMPONENT_ENGINE component for each GPU engine (node) A DXGK_POWER_COMPONENT_MEMORY component for every memory segment and with the DXGK_POWER_COMPONENT_FLAGS::ActiveInD3 flag set. This component must report 2 F States and TransitionalLatency of F1 state must be 0 One DXGK_POWER_COMPONENT_MEMORY_REFRESH component for the adapter. Also, the driver must leave space in dependency array for all device engines

- Transitional Latency reported for each component must not be greater than max. Latency tolerance for that component is specified in the table below.

	Latency tolerance

Engine (monitor ON) Initial state After 200 ms of idle time No context on the engine	. 0.08 ms 15 ms 30 ms
Engine (monitor OFF) Initial state After 200 ms of idle time No context on the engine	. 2 ms 50 ms 100 ms
Memory Active context exists No active context exists	. 15 ms 30 ms
Memory refresh Initial state No active context exists Monitor off and no active context exists	. 0.08 ms 30 ms 80 ms
D3 transition Initial state After 10 s of all engines idle time No active context Monitor off and (no active context or all engines idle for 60 s)	. 0.08 ms 15 ms 200 ms 250 ms

Send comments about this topic to Microsoft

System.Fundamentals.Graphics.InternalDisplay

Base for Graphics on Systems

In this topic:

System.Fundamentals.Graphics.InternalDisplay.NativeResolution

Systems with integrated displays must use native resolution by default.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A system with an integrated display must support the native resolution of the display and use native resolution as the default.

An "integrated" display is any display that is built into the system. A laptop lid is an example of an integrated display.

Windows is designed to work best in native resolution.

This requirement applies to systems that use UEFI or BIOS.

Send comments about this topic to Microsoft

System.Fundamentals.Graphics.MultipleDevice

Requirements which apply to systems with more than one graphics device.

In this topic:

System.Fundamentals.Graphics.MultipleDevice.Configure

On a system with multiple graphics adapters, system firmware will allow the user to configure the usage of the adapters.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

On a system with multiple graphics adapters, the system firmware (BIOS, UEFI, etc), must provide the user with the ability to modify the following settings:

- Enable/Disable any adapters:

- The firmware must offer the user the ability to select which adapter is enabled or disabled
- At any given time at least one adapter, that supports POST, must be enabled
- If the user enables an adapter, and the system only supports one active adapter at a time, then all other adapters must be disabled
- If the only enabled adapter is not detected, the firmware will, fallback to the integrated adapter. If there is no integrated adapter, then fallback to the first adapter found on the first bus

- Select the adapter to be used as POST device
  - Firmware must only allow the user to select one adapter as the POST device.
  - A System with an integrated adapter is allowed to POST only on an adapter that cannot be physically removed from the system
  - At any given time at least one adapter, that supports POST, must be enabled
  - If multiple adapters that support POST are enabled, the firmware must provide the user an option to select which one will be used for POST

System.Fundamentals.Graphics.MultipleDevice.SubsystemDeviceID

Hybrid/Switchable Graphics systems that support multiple discrete graphics adapters or chipset combination must use the same Subsystem ID.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Multiple GPU Graphics configurations that support multiple discrete graphics adapters or chipset combination must use the same Subsystem ID for each device in the configuration.

Multiple GPU Graphics systems are permitted to have heterogeneous graphics solutions in certain circumstances, thus allowing different VEN IDs.

The Multiple GPU configurations which combine GPUS from different vendors must have the same SUBSYS ID to indicate the driver packages intended for a Multiple GPU system. Should the same device be used as a single device in another system, that instance of the device must use a different unique 4part PNPid.

This does not apply to systems that implement Microsoft Hybrid solution, this solution is expected to have distinct HWID's.

Examples:

1. The integrated GPU and the Discrete GPU may have different VEN ID and DEV ID, but must have the same SSID. For example:

Display Devices

Card name: InField GFX

Manufacturer: OutStanding

Chip type: RUOK Family

DAC type: Integrated RAMDAC

Device Key: Enum\PCI\VEN_AAAA&DEV_EEEE&SUBSYS_9025104D&REV_A1

Display Devices

Card name: Rocking Fast GFX

Manufacturer: Awesome Chips

Chip type: 10Q Family

DAC type: Internal

Device Key: Enum\PCI\VEN_BBBB&DEV_DDDD&SUBSYS_9025104D&REV_07

2. The GPUs that is used in a Switchable machine must use a different SSID if also used in a non-switchable machine. For example:

Display Devices

Card name: InField GFX

Manufacturer: OutStanding

Chip type: RUOK Family

DAC type: Integrated RAMDAC

Device Key: Enum\PCI\VEN_AAAA&DEV_EEEE&SUBSYS_9999104D&REV_A1

Note that the OutStanding InField GFX in #1. Is the same as the one stated in #2; however, although they are the same hardware, they must have a different SSID.

Send comments about this topic to Microsoft

System.Fundamentals.Graphics.RenderOnly

Requirements which apply to a graphics device only implementing WDDM Render DDI's.

In this topic:

System.Fundamentals.Graphics.RenderOnly.MinimumDirectXLevel

Render Only device on client or server system must be Direct3D 10 capable or greater.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If a client or server system includes a render only device, the device must be Direct3D 10 capable or greater. This device can only be supported by a WDDMv1.2 Render Only Driver. Render Only devices are not allowed as the primary graphics device on client systems. All Windows client systems must have a full graphics WDDM v1.3 device as the primary boot device.

Send comments about this topic to Microsoft

System.Fundamentals.HAL

This feature defines Hardware Abstraction Layer (HAL) requirements for systems.

In this topic:

System.Fundamentals.HAL.HPETRequired

System provides a high-precision event timer

Applies to	Windows Server 2016 Technical Preview x64

Description

Systems must implement a High Precision Event Timer (HPET) that complies with the following points of the Intel Architecture Personal Computer (IA-PC) HPET Specification:

- The main counter frequency must be greater than or equal to 10 MHz and less than or equal to 500 MHz
- The main counter must monotonically increase, except on a roll-over event.
- The main counter and comparators must be at least 32 bits wide.
- The main counter must have at least three comparators.
- All of the comparators must be able to fire aperiodic, "one-shot" interrupts.
- At least one of the comparators must be able to fire periodic interrupts.
- Each comparator must be able to fire a unique and independent interrupt.
- HPET must support edge triggering interrupts.
- Timer interrupts must not be shared in LegacyIRQRouting mode.

Send comments about this topic to Microsoft

System.Fundamentals.Input

Requirements in this section apply to HID devices that are integrated in the system.

In this topic:

System.Fundamentals.Input.I2CDeviceUniqueHWID

I2C connected HID devices must have a Unique HWID along with a HID compatible ID.

Applies to	Windows 10 x64 Windows 10 x86

Description

All I2C connected HID devices must have a unique HWID and a HID compatible ID that will allow WU to identify the device (when needed) and allow drivers to be loaded from WU.

Design Notes:

See Microsoft published HID I2C protocol specification.

System.Fundamentals.Input.PS2UniqueHWID

All PS/2 connected devices (such as internal keyboards) must have a unique hardware ID.

Applies to	Windows 10 x64 Windows 10 x86

Description

All PS/2 connected devices (such as touchpads and keyboards) must have a unique hardware ID that enables the third party driver to ship with WU.

Design Notes:

See Microsoft unique hardware ID whitepaper http://www.microsoft.com/whdc/device/input/mobileHW-IDs.mspx.

Send comments about this topic to Microsoft

System.Fundamentals.MarkerFile

A marker file is used to help associate WER data with specific computer models. Requirements in this section describe the syntax for the "marker file.

In this topic:

System.Fundamentals.MarkerFile.SystemIncludesMarkerFile

System includes marker file

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The marker file gives additional information regarding the maker of the PC system and model. This information is used to collect and distribute On-line Crash Analysis information. The marker file is a text file with a .mrk extension. The .MRK filename must be under 256 characters in length including the path. The characters must be letters, numbers, periods, hyphens, commas and parentheses.

The marker file format is:

For companies with PCI Vendor IDs:

VendorID_CompanyName_Division_Marketing Model Name_other info.MRK

For companies without a PCI Vendor ID

CompanyName_Division_Marketing Model Name_other info.MRK

Each column is separated by the underscore '_' character. The values in each column are

VendorID = The PCI vendor ID for the PC manufacturer.

CompanyName = Name of the company go here. This should be consistent for each marker file.

Division = this represents the division within the company. If your company doesn't not have divisions please put 'na.'

Marketing Model Name = product name the system will be shipped as. This should be the same as the marketing name entered at the time of logo submission.

Other info = optional ad can be added by putting more underscores. The additional fields may be used for identifying any other critical information about the system.

Optionally, the _I field can be used as a part number that can be used to link the marketing model name to.

Design Notes:

The marker file goes in the c:\windows\system32\drivers folder.

Send comments about this topic to Microsoft

System.Fundamentals.Network

These are system level requirements that may impact the integration with a type of network device.

In this topic:

System.Fundamentals.Network.NetworkListOffloads

Wireless LAN networking device on systems that support Connected Standby must support NDIS 6.30 and support offloads.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following requirements apply to wireless LAN devices. WLAN Devices must support the following features.

Feature	Requirement
No Pause on Suspend	Required
D0 offload	Required
USB Selective Suspend	Required- If USB based
Network List offload	Required
Wi-Fi PSM	Required
Wi-Fi Direct	Required
Radio Management	Required
WPS 2.0	Required
WoWLAN	Required

Systems that support Connected Standby require the use of an NDIS 6.30 Ethernet driver. The device must support the features listed below.

Feature	Required
Wakeup on LAN	Yes
D0 & D3 Protocol Offloads (Protocols Jun. 26, 2013)	Yes
Interrupt Moderation	Yes
OS-programmable packet filtering	Yes

System.Fundamentals.Network.PowerRequirements

All physical network devices in a system (inclusive of docking stations) must meet device certification criteria for power management requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Support of this feature is required. All physical network devices included in a system (inclusive of docking stations) must meet the device-level power management requirements for that specific device type. Example: If an Ethernet device is included in a Connected Standby capable system or associated dock, that Ethernet device must meet the power management requirements for Connected Standby regardless of whether the individual device certification was achieved when tested on a Connected Standby capable system or not.

Send comments about this topic to Microsoft

System.Fundamentals.NX

In this topic:

System.Fundamentals.NX.SystemIncludesNXProcessor

Systems must ship with processors that support NX and include drivers that function normally when NX is enabled

Applies to	Windows Server 2016 Technical Preview x64

Description

To ensure proper device and driver behavior in systems all drivers must operate normally with Execution Protection. Specifically, drivers must not execute code out of the stack, paged pool and session pool. Additionally, drivers must not fail to load when Physical Address Extension (PAE) mode is enabled, a requirement for operation of NX.In addition, the system firmware must have NX on and data execution prevention (DEP) policy must not be set to “always off."

Send comments about this topic to Microsoft

System.Fundamentals.PowerManagement

Power management is a feature that turns the PC off or into a lower power state. Requirements in this section describes requirements around power management.

In this topic:

System.Fundamentals.PowerManagement.DockUndock

System supports docking and undocking across a hibernate transition.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

For systems which ship with a dock, the system must be able to hibernate and resume when changing from the docked to undocked state or the undocked to the docked state. This is not limited to, but should include that the memory map should not change when docking or undocking the system.

System.Fundamentals.PowerManagement.MultiPhaseResume

Storage subsystem supports multi-phase resume from Hibernate

Applies to	Windows 10 x64 Windows 10 x86

Description

The driver and hardware subsystems for the boot storage device must support multi-phase resume from Hibernate. In order to do this, the system must be able to maintain the system's ability to identify definitively all of the memory needed on resume. This is not limited to, but should include that:

- Any crashdump filters/minifilters that must support read
- No WHEA pshed plugins are installed

- Hypervisor is not enabled

System.Fundamentals.PowerManagement.PCSupportsLowPowerStates

Systems support S4 and S5 and either S0 low power idle or S3, states.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A desktop or mobile system installed with a client operating system must support the S4 (Hibernate) and S5 (Soft-off) states and either S0 low power idle, or S3 (Sleep). Systems that support Connected Standby must also support S4 (Hibernate). Every system must support wake from all implemented sleep states. Wake from S5 is only required from the power button.

Systems which support S0 low power idle must report that behavior by setting the following bits in the FACP flags.

FACP - Field	Bit Len.	Bit Offset	**Description**
LOW_POWER_S0_IDLE_CAPABLE	1	21	This flag indicates if the system supports low power idle states in the ACPI S0 state. A value of one (1) indicates that the platform supports sufficiently low S0 idle power such that transitions to the S3 state are not required. OSPM may interpret a one in a manner that it favors leaving the platform in the S0 state with many devices powered off over the S3 state when the user is no longer interacting with the platform.
Reserved	10	22	Reserved for future use.

If a USB host controller is implemented on the system, then at least one external port on the controller must support wake-up capabilities from S3. If the system contains multiple USB host controllers, all host controllers integrated on the system board (that is, not add-on cards) must support wake-up from S3. USB host controllers are not required to support wake-up when a mobile system is running on battery power.

Server systems are not required to implement S0 idle, S3, S4, or S5 states. If a server system does implement any of these behaviors, they must work correctly.

Power Management is an important aspect of good user experience. The system should be able to control what devices to put into a sleep state when not being used. All devices must comply with the request from the system to go into a sleep state and not veto the request thereby putting an additional drain on the power source.

System.Fundamentals.PowerManagement.PowerProfile

System must report form factor via power management profile.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The Preferred_PM_Profile in the FADT table must be set to one of the values based on the form factor of the system as outlined in the ACPI specification version 5.0. This value shall not be unspecified (0).

Design Notes:

For more information see page 119 of the ACPI specification version 5.0.

Send comments about this topic to Microsoft

System.Fundamentals.PowerManagement.CS

Power management is a feature that turns the PC off or into a lower power state. Requirements in this section describes requirements around power management for systems that support connected standby.

In this topic:

System.Fundamentals.PowerManagement.CS.CSQuality

Systems that support S0 low power idle must meet reliability standards for Runtime Power Management.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems that support Connected Standby must meet minimal reliability standards as tested for this requirement. The test associated with this requirement will exercise any installed Power-Engine Plug-In (PEP), installed device drivers and platform firmware.

Design Notes:

To help ensure the reliability of a system that supports connected standby, the system will be subjected to the following tests:

- Connected Standby Stress with IO
- Runtime Power Focused Stress with IO

These tests will be run while Driver Verifier is enabled with standard settings.

These tests will also be run separately with the Driver Verifier Concurrency Testing setting.

If a PEP device is enumerated in ACPI namespace and the system does not have a PEP loaded, the test will fail.

Send comments about this topic to Microsoft

System.Fundamentals.PXE

In this topic:

System.Fundamentals.PXE.PXEBoot

Remote boot support for PXE complies with BIOS Boot Specification 1.01 or EFI boot manager

Applies to	Windows Server 2016 Technical Preview x64

Description

All systems are required to be PXE capable. The system must support booting from the network as defined in BIOS Boot Specification, Version 1.01, Appendix C, or as controlled by the EFI boot manager. This requirement is exempt for systems that are configured with Wireless LAN only. Systems shipping with a UEFI compatible operating system and supporting PXE boot must support IPV4 PXE and IPV6 PXE booting as defined in UEFI 2.3.1.

UNDI must support:

- a DUID-UUID per IEFT draft
- http://tools.ietf.org/html/draft-narten-dhc-duid-uuid-01

- DHCP6, DUID-UUID, IPv6 IPV4 multicast

Design Notes:

Microsoft recommends that the implementation of accessing PXE be consistent with BIOS Boot Specification, Version 1.01, and Appendix C.

Send comments about this topic to Microsoft

System.Fundamentals.Reliability

In this topic:

System.Fundamentals.Reliability.SystemReliability

Drivers in a system must be reliable.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All drivers in a system must pass all requirements under **Device.DevFund.Reliability**. All systems will need to pass Common Scenario stress:

- Enable/Disable with IO before and after
- Sleep stress with IO before and after

Send comments about this topic to Microsoft

System.Fundamentals.Security

In this topic:

System.Fundamentals.Security.DeviceEncryption

Systems that support connected standby must support device encryption.

Applies to	Windows 10 x64 Windows 10 x86

Description

Systems that support connected standby must meet the security requirements to support enablement of Device Encryption. OEMs must not block the enablement of Device Encryption when deploying the OS images unless the device is pre-provisioned with a third-party disk encryption solution. Device Encryption will be enabled on these systems to ensure that user data is protected. As pre-requisites for Device Encryption, connected standby systems must meet requirements for TPM and Secure Boot as outlined in System.Fundamentals.TPM20 and System.Fundamentals.Firmware.CS.UEFISecureBoot.ConnectedStandby.

System.Fundamentals.Security.NoTDIFilterAndLSP

No TDI filters or LSPs are installed by the driver or associated software packages during installation or usage.

Applies to	Windows 10 x64 Windows 10 x86

Description

There can be no use of TDI filters or LSPs by either kernel mode software or drivers, or user mode software or drivers.

System.Fundamentals.Security.PlayReadyModule

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

PlayReadyModule, when available on a device in secure firmware in conjunction with a compatible graphics driver, enables hardware-based content protection for media. If implemented, this module provides hardware-rooted protection of device keys, content keys and media content/samples that flow through a media pipeline. It will enable the device to have access to high definition (1080p and above) premium content. OEMs shipping on chipsets/SoCs that have a PlayReadyModule available (in the form of secure firmware available from the chipset vendor) must include PlayReadyModule on devices with screen resolutions of 1080p or higher.

Send comments about this topic to Microsoft

System.Fundamentals.SignedDrivers

This feature checks for signed drivers.

In this topic:

System.Fundamentals.SignedDrivers.BootDriverEmbeddedSignature

Boot drivers must be self-signed with an embedded signature.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All boot start drivers must be embedded-signed using a Software Publisher Certificate (SPC) from a commercial certificate authority. The SPC must be valid for kernel modules. Drivers must be embedded-signed through self-signing before the driver submission.

Design Notes:

For more information about how to embedded-sign a boot start driver, see "Step 6: Release-Sign a Driver Image File by Using an Embedded Signature" at the following website:

http://www.microsoft.com/whdc/winlogo/drvsign/kmcs_walkthrough.mspx

After the file is embedded-signed, use SignTool to verify the signature. Check the results to verify that the root of the SPC's certificate chain for kernel policy is "Microsoft Code Verification Root." The following command line verifies the signature on the toaster.sys file:

Signtool verify /kp /v amd64\toaster.sys

Verifying: toaster.sys

SHA1 hash of file: 2C830C20CF15FCF0AC0A4A04337736987C8ACBE3

Signing Certificate Chain:

Issued to: Microsoft Code Verification Root

Issued by: Microsoft Code Verification Root

Expires: 11/1/2025 5:54:03 AM

SHA1 hash: 8FBE4D070EF8AB1BCCAF2A9D5CCAE7282A2C66B3

...

Successfully verified: toaster.sys

Number of files successfully Verified: 1

Number of warnings: 0

Number of errors: 0

In the Windows Hardware Lab Kit, this requirement will be tested by using the Embedded Signature Verification test.

System.Fundamentals.SignedDrivers.DigitalSignature

System must contain compatible qualified devices.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All buses, devices and other components in a system must meet their respective Windows Compatible Hardware Requirements and use drivers that either are included with the Windows operating system installation media or are digitally signed by Microsoft through the Windows Hardware Compatibility Program that match the Windows OS version being submitted for and shipping with.

For example, if a logo qualifying a system for Windows 10, then all drivers on the system must be signed by Microsoft for Windows 10 or be drivers that ship on the Windows 10 media. All devices in the system would also need to be logo qualified or certified for Windows 10. This requirement applies to all versions of Microsoft Windows.

All devices and drivers need to be fully installed, and does not contain any problem codes.

Send comments about this topic to Microsoft

System.Fundamentals.SMBIOS

System Management BIOS (SMBIOS) requirements defines data structures in the system firmware which allows a user or application to store and retrieve information about the computer.

In this topic:

System.Fundamentals.SMBIOS.SMBIOSSpecification

System firmware support for SMBIOS complies with the SMBIOS specification.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The system firmware must implement support for SMBIOS that complies with System Management BIOS Reference Specification, Version 2.4 or later. The SMBIOS implementation must follow all conventions and include all required structures and fields as indicated in the SMBIOS Specification, Section 3.2, and follow all conformance requirements as indicated in Section 4. Bit 2 in the BIOS Characteristics Extension Byte 2 field must be set (Section 3.3.1.2.2 of the specification). The length of the Type 1 (System Information) table must be at least 1Bh bytes (includes SKU Number and Family fields from Version 2.4 of the specification).

Additionally, the following fields must have non-Null values that accurately describe the computer system or computer system component:

- (Table 0, offset 04h) BIOS Vendor
- (Table 0, offset 08h) BIOS Release Date
- (Table 0, offset 14h) BIOS Major Release Version[1]
- (Table 0, offset 15h) BIOS Minor Release Version[1]
- (Table 1, offset 04h) System Manufacturer[2]
- (Table 1, offset 05h) System Product Name[2]
- (Table 1, offset 08h) Universal Unique ID number
- (Table 1, offset 19h) System SKU Number[2]

Microsoft recommends that the following fields have non-Null values that accurately describe the computer system or computer system component:

- (Table 0, offset 05h) BIOS Version
- (Table 0, offset 16h)Embedded Controller Major Release Version[3]
- (Table 0, offset 17h) Embedded Controller Minor Release Version[3]
- (Table 1, offset 06h) System Version
- (Table 1, offset 1Bh) System Family[2]

- (Table 2, offset 04h) Base Board Manufacturer
- (Table 2, offset 05h) Base Board Product
- (Table 2, offset 06h) Base Board Version

[1]These fields must not have values equal to 0FFh.

[2]These fields gain prominence as fields which will be used for identifying unique system configurations for telemetry and servicing. The Manufacturer, Product Name, SKU Number and Family fields must not be longer than 64 characters in length. Avoid leading or trailing spaces or other invisible characters.

[3]If the system has a field upgradeable embedded controller firmware; these values should not be equal to 0FFh.

Design Notes: SKU Number has been moved to a required field in order to improve telemetry reporting. We encourage the OEM to be careful to fill in Manufacturer consistently and to fill in SKU Number with a value that can identify what the OEM considers a unique system configuration for telemetry and servicing.

Send comments about this topic to Microsoft

System.Fundamentals.StorageAndBoot

This section summarizes the requirements for storage and boot devices.

In this topic:

System.Fundamentals.StorageAndBoot.BootPerformance

Boot Devices in systems that support Connected Standby must meet these requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following requirements apply to Boot Devices in systems that support Connected Standby.

In addition to SATA and USB, Windows platform supports SD and eMMC based storage. An eMMC device may be used as either a system (boot) disk or as a data disk but an SD device can only be used as a data disk.

Flash Type	Boot	Data Disk
eMMC 4.5+	Yes	Yes
SD 2.0 or 3.0	No	Yes

ACPI interfaces must specify whether the storage device is internal or external and whether or not it is removable or fixed.

_RMV must be defined in ACPI namespace for any embedded devices attached to an SD host controller, where 0 is defined as non-removable. _RMV may optionally be defined for external slots as 1.

The following parameters must be defined within the “Storage Class-Specific Information” to be returned by the ACPI _DSM method for an SD/eMMC Storage Controller:

- Number of Sockets
- Socket Addresses

Support GPIO card detection on SD/eMMC Storage Controller.

When using eMMC as the primary boot device, the eMMC memory must be hardware partitioned such that the boot critical portion of the EFI Firmware resides in an area of the device that is not accessible by Windows.

The CPU Vendor and/or Firmware Provider must furnish the software tools needed to maintain and update the firmware.

The following requirements are applicable to boot storage media and are tested with the smaller of 2% or 1GB free space.

Feature	Span Size	Specification
Power		
Max Idle Power	-	<= 5 mW
Random Performance		
4KB Write IOPs	1 GB	>= 200
	*5 GB	>= 50

	†10 GB	>= 50
64KB Write IOPs	1 GB	>= 25
4KB Read IOPs	*5 GB	>= 2000
	†10 GB	>= 2000
4KB 2:1 read/write mix IOPs	1 GB	>= 500
	* 5GB	>= 140
	†10 GB	>= 140
Sequential Performance		
Write speed (64KB I/Os)	*5 GB	>= 40 MB/s
	†10 GB	>= 40 MB/s
Write speed (1MB I/Os)	*5 GB	>= 40 MB/s
	†10 GB	>= 40 MB/s
Read speed (64KB I/Os)	*5 GB	>= 60 MB/s
		‡>= (120 MB/s)
	†10GB	>= 60 MB/s
		‡>= (120 MB/s)
Device I/O Latency		
Max Latency	-	< 500 milliseconds

*Applies only to devices with 16 GB of internal storage or lower.

†Applies only to devices with greater than 16 GB of internal storage.

‡Applies only if the device is HS200 capable.

Additional I/O Latency requirement:

Maximum of **20 seconds** sum-total of user-perceivable I/O latencies over any **1 hour** period of a user-representative workload, where a user-perceivable I/O is defined as having a latency of at least 100 milliseconds.

System.Fundamentals.StorageAndBoot.EncryptedDrive

Systems which ship with a Encrypted Drive as a boot storage device must support security command protocols in order to make sure the data at rest is always protected.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

The following requirements apply if Encrypted Drive (**System.Fundamentals.StorageAndBoot.EncryptedDrive**) support is implemented for a UEFI based client system or server systems:

- The system MUST have a TPM 1.2 or TPM 2.0.
- The system MUST implement the EFI_STORAGE_SECURITY_COMMAND_PROTOCOL from either:
  - Trusted Command Support in UEFI 2.3 + UEFI Mantis change number 616 or
  - UEFI 2.3.1
- The implementation of the Trusted Computing Group Platform Reset Attack Mitigation Specification (http://www.trustedcomputinggroup.org/resources/pc_client_work_group_platform_reset_attack_mitigation_specification_version_10) MUST unconditionally issue TPer Reset (OPAL v2.0 in section 3.2.3) for all scenarios whenever memory is cleared.
- The EFI_STORAGE_SECURITY_COMMAND_PROTOCOL and the TPer Reset command MUST be included in the base UEFI image (not in a separate image of a UEFI driver).
- The system MUST enumerate all Encrypted Drives and TPer Reset MUST be issued prior to executing any firmware code not provided by the platform manufacturer in the base UEFI image.
- The TPer Reset MUST be issued regardless of whether the TPM has had ownership taken or not.

Note: The TPer Reset action will occur later in the boot process than the memory clear action because it has a dependency on the EFI_STORAGE_SECURITY_COMMAND_PROTOCOL.

System.Fundamentals.StorageAndBoot.SATABootStorage

System with SATA boot storage must meet requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

AHCI Host Controllers using Windows for boot support must be compliant with the AHCI specification.

Host Controller Interface	Revision
AHCI	1.1, 1.2, 1.3

System with SATA controller must enable AHCI mode support.

Externally connected SATA devices (eSATA) are not supported for boot storage.

When SATA is used as the primary boot device, to ensure reliability and prevent inadvertent erasure of the firmware that may cause the device to become inoperable, the boot critical portion of the UEFI firmware must reside on a separate storage device that is not accessible by the host Operating System. The CPU Vendor and/or Firmware Provider must furnish the software tools needed to maintain and update the firmware.

When used in systems that support connected standby, the SATA device must meet the power requirements stated in the section for **System.Fundamentals.StorageAndBoot.BootPerformance**.

Send comments about this topic to Microsoft

System.Fundamentals.SystemAudio

In this topic:

System.Fundamentals.SystemAudio.Audio

Systems contain audio devices that conform to Windows Logo requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems need to conform to all **Device.Audio** requirements.

System.Fundamentals.SystemAudio.MicrophoneLocation

Microphone Location Reporting Requirement

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

- All active onboard fixed-position single element microphones and onboard fixed-position microphone arrays (multiple combined elements) on a system must be available for independent capture.
- Systems with no onboard fixed-position microphone arrays, but with multiple onboard fixed-position microphones (e.g. Front/Back), AND no combined microphone, must have exactly one with GeoLocation = “Front” (which will be set as the default microphone on that system).
- Systems with multiple onboard fixed-position microphone arrays must have exactly one with GeoLocation = “Front.”
- Mic arrays with “n” elements must deliver RAW audio to the system. The RAW format must have “n” channels, and data must come directly from the mic elements, free of signal processing.
- If a microphone and a camera are physically co-located onboard then stated location information for each must match.

- For devices with multiple onboard fixed-position microphones or multiple arrays, the names of these endpoints should be unique on the system. To specify a unique name, there are a few different methods using KSPROPERTY_PIN_NAME, IPinName and .inf pin description name GUID registration.

System.Fundamentals.SystemAudio.SystemUsesHDAudioPinConfigs

System uses the HD Audio device pin configuration registers to expose logical devices supported by the Windows UAA HD Audio class driver.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

If the PnP ID of an HD Audio device matches as compatible with any of the audio class drivers packaged with Windows, the device must provide basic functionality for all of its endpoints when using that driver. Audio device must follow Microsoft HD Audio pin configuration programming guidelines and expose devices divided into areas based on audio device hardware functionality resources.

See the Pin Configuration Guidelines for High Definition Audio Devices white paper at http://go.microsoft.com/fwlink/?LinkId=58572.

Send comments about this topic to Microsoft

System.Fundamentals.SystemPCIController

In this topic:

System.Fundamentals.SystemPCIController.PCIRequirements

System devices and firmware meet PCI requirements

Applies to	Windows Server 2016 Technical Preview x64

Description

All PCI Express devices must comply with the PCI Base Express Specification 1.1 or later unless specified otherwise below.

Reverse bridge implementations as defined in Appendix A of the PCI Express to PCI/PCI-X Bridge Specification are not supported in Windows. A reverse bridge will not be supported if it adheres to the guidelines and recommendations as defined in Appendix A of the PCI Express to PCI/PCI-X Bridge Specification, Revision 1.0.

System firmware disables the extended (non-VC0) virtual channels in PCI Express devices. The system firmware sets the VC enable bit (PCI Express Base Specification, Revision 1.0a, Section 7.11.7, "VC Resource Control Register: bit 31") to 0 for all extended (non-VC0) virtual channels in all PCI Express devices. This requirement does not apply to PCI Express High Definition Audio controllers, which use class code 04 and subclass code 03.Because extended support for VC hardware is optional, this requirement addresses the scenario in which incompatible VC hardware implementations might cause system reliability, stability, and performance issues. Hardware vendors are encouraged to work with Microsoft to define the future direction of extended virtual channel support.

System firmware for PCI-X Mode 2 capable and PCI Express systems implements MCFG table for configuration space access. PCI-X Mode 2-capable and PCI Express systems must implement the MCFG ACPI table in PCI Firmware. Specification, Revision 3.0.The configuration space of PCI-X Mode 2 and PCI Express devices must be accessible through the memory-mapped configuration space region defined in this table.

PCI-to-PCI bridges comply with PCI-to-PCI Bridge Architecture Specification

All PCI-to-PCI bridges must comply with PCI-to-PCI Bridge Architecture Specification, Revision 1.2.

Virtual bridges that comply with PCI Express also comply with PCI-to-PCI Bridge Architecture Specification, Revision 1.1. In addition, VGA 16-bit decode (Section 3.2.5.18, "Bridge Control Register, bit 4") and SSID and SSVID (Section 3.2.5.13) from PCI-to-PCI Bridge Architecture Specification, Revision 1.2, must also be supported. SSID and SSVID support is not required until January 1, 2011. If implemented, SSID and SSVID must meet the specification. SSVID is not required for PCIe to PCI/PCI-X bridges.

x64-based system provides 64-bit support in PCI subsystem. For x64-based systems, all PCI bridges on the system board must support dual-access cycle (DAC) for inbound access, and DAC-capable devices must not be connected below non-DAC-capable bridges, such as on adapter cards.

All 64-bit adapters must be DAC capable. This DAC requirement does not apply to outbound accesses to PCI devices. However, for systems in which DAC is not supported on outbound accesses to PCI devices, the system firmware must not claim that the bus aperture can be placed above the 4-GB boundary.

Send comments about this topic to Microsoft

System.Fundamentals.SystemUSB

This section contains requirements for systems with USB host controllers.

In this topic:

- System.Fundamentals.SystemUSB.SystemExposesUSBPort
- System.Fundamentals.SystemUSB.TestedUsingMicrosoftUsbStack
- System.Fundamentals.SystemUSB.USBDevicesandHostControllersWorkAfterPowerCycle
- System.Fundamentals.SystemUSB.XhciBiosHandoffFollowsSpec
- System.Fundamentals.SystemUSB.XHCIControllersMustHaveEmbeddedInfo
- System.Fundamentals.SystemUSB.XhciSupportsMinimum31Streams

System.Fundamentals.SystemUSB.ExternalUSBonCSisEHCIorXHCI

External USB ports on system that support connected standby must be EHCI or XHCI

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

USB host controllers on systems that support Connected Standby must implement xHCI (eXtensible Host Controller Interface) or EHCI (Enhanced Host Controller Interface). Legacy companion controllers (UHCI/OHCI) are not supported.

All exposed ports must support all speeds slower than the maximum speed of the host controller, to enable support of legacy devices including keyboards and mice.

Required Speed Support	EHCI Port (USB 2.0)	XHCI Port (USB 3.x)
Low-Speed	Yes	Yes
Full-Speed	Yes	Yes
Hi-Speed	Yes	Yes
Super-Speed	No	Yes

Transaction translators (TTs), integrated with the EHCI host controller, are not standardized, but the Windows EHCI driver supports serveral implementations of a controller- integrated TT. The supported integrated TT implementation must be identified in ACPI using the _HRV hardware revision for the USB controller. Please contact the USB team to determine if your implementation is supported and for more information about which _HRV value should be reported.

If the USB EHCI controller does not feature an integrated TT, any externally exposed ports must be routed through an embedded rate-matching hub.

For improved power efficiency and performance, USB Host Controllers on systems that support Connected Standby are recommended to be at least USB 3.x compatible, with an XHCI controller integrated into the SoC or chipset. The operating system supports standard EHCI and XHCI controllers including debug registers.

USB Host Controller Interface	Recommendation
UHCI/OHCI Companion Controllers	Not-supported
EHCI	Supported
XHCI (including debug capability)	Supported and Recommended

System.Fundamentals.SystemUSB.SuperSpeedCapableConnectorRequirements

Each exposed SuperSpeed capable connector supports SuperSpeed, high, full and low speed USB devices routed through its xHCI controller.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

xHCI Controllers are backwards compatible with SuperSpeed, high, full, and low speed USB devices. Backwards compatible is defined as all USB devices enumerate and function at their intended speeds. More than one xHCI controller may be present on a system as long as the SuperSpeed capable ports are correctly routed. EHCI controllers may also be present on the system; however, SuperSpeed capable ports should not be routed through them.

System.Fundamentals.SystemUSB.SystemExposesUSBPort

Systems are recommended to expose at least one user-accessible USB port.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems are recommended to expose at least one external USB host port. If a system exposes such port, the following requirement applies:

For maximum capabilities with Windows, systems that expose such ports are recommended to have an externally accessible standard USB A port and a USB Type-C port. This allows users to connect their existing USB devices and new Type-C devices without an adapter.

If the system's form factor is too thin to expose a standard USB A port, it is recommended to expose a Type-C port and acceptable to expose a micro-A/B port. See the table below for the complete list of options.

While USB 2.0 capable controllers are acceptable, USB 3.1 XHCI host controllers are preferred. The USB ports must fully comply with the USB 3.1 or USB 2.0 specification. USB 3.1 connectors must properly support 900mA USB 3.1 devices, and 500 mA USB 2.0 and 1.1 devices. USB 2.0 ports must properly support 500 mA USB 2.0 and 1.1 devices.

It is optional to include an adapter that converts the port from micro USB or USB Type-C to USB A. If you bundle an adapter, the adapter capabilities must match that of the exposed connector of the USB host controller.

External USB Ports	Recommendation
Standard USB A Port(s)	Recommended
USB Type-C	Recommended
Standard USB A Port(s) + USB Type-C Port(s)	Recommended (Preferred)
Micro-USB A/B Port + 1 or more Standard USB A Port	Supported
Micro-USB A/B (Host + Function debug) Port	Supported
Micro-USB B Port + 1 or more Standard USB A Port	Supported
Proprietary Docking Port with USB Host and/or debug Functionality	Supported
Mini-USB A, A/B or B Port	Not Supported
Proprietary USB Host Port	Not Supported

Whatever USB port type is chosen, it must be correctly described in ACPI with the _UPC (USB Port Capabilities) package type parameter as defined in the ACPI 4.0a specification, section 9.13. This

information allows Windows to determine when a micro-A/B port is exposed, and ID pin detection is necessary.

A simple Standard USB A male to Micro USB B female adapter can be used to expose USB function or XHCI host debug transport from a Standard USB-A port. This adapter must prevent shorts on the VBus line by removing the VBus line completely or by having a 1kOhm resistor in line with the VBus line. It is strongly recommended that the standard USB A port provide built-in protection against a short on the VBus line. This can occur if the USB port is connected to another host when it is not properly configured in debug mode.

If a system exposes multiple Dual Role capable ports, only one port should in function mode at any given time. If the micro-USB B port provides no additional functionality beyond debugging, it must be hidden in the battery compartment or behind a easily removable cover. In order to comply with USB-IF requirements, VBUS must not be asserted on the micro-A/B port until the resistance to ground of the ID pin of the micro-USB A/B port is less than 10 Ohms. This will prevent a short-circuit when a user connects a micro-USB B cable to another USB host, such as a desktop. Alternatively, the port can implement short protection circuitry for VBus.

System.Fundamentals.SystemUSB.TestedUsingMicrosoftUsbStack

Systems with xHCI Controllers must be tested with Microsoft's xHCI Stack installed.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Systems with Extensible Host Controller Interface (xHCI) Controllers must be tested with Microsoft's xHCI Stack installed and enabled.

System.Fundamentals.SystemUSB.USBDevicesandHostControllersWorkAfterPowerCycle

All USB devices and host controllers work properly upon resume from sleep, hibernation or restart without a forced reset of the USB host controller.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All USB devices and host controllers work properly upon resume from sleep, hibernation or restart without a forced reset of the USB host controller.

Design Notes:

Registry key ForceHCResetOnResume documented at the KB below is not needed for devices to function properly upon resume in Windows 7 or newer.

http://support.microsoft.com/kb/928631

Note that a known set of currently existing devices do require a forced reset upon resume, these devices should be covered in a list kept by the OS which will reset these devices upon resume. The goal of this requirement is to ensure that this list of devices which need a reset to appear after resume does not grow and that devices can properly handle sleep state transitions without being reset.

A reset of the entire USB Host Controller results in significantly increased time that it takes for all USB devices to become available after system resume since there could be only one device at address 0 at a time, this enumeration has to be serialized for all USB devices on the bus. We have also seen that resetting the host controller can lead to an illegal SE1 signal state on some host controllers, which in turn can cause some USB devices to hang or drop off the bus. Moreover, devices cannot maintain any private state across sleep resume as that state will be lost on reset.

System.Fundamentals.SystemUSB.XhciBiosHandoffFollowsSpec

xHCI BIOS handoff follows specification

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

For all xHCI controllers exposed to the OS, the system firmware must follow the BIOS handoff procedure defined in section 4.2.2.1 of the XHCI specification.

System.Fundamentals.SystemUSB.XHCIControllersMustHaveEmbeddedInfo

Systems with xHCI controllers must have embedded ACPI information for port routing.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Please reference ACPI specification version 4.0.

The ACPI namespace hierarchy for USB devices should exactly match the devices hierarchy enumerated by Windows operating system.

All connectable USB ports are required to have a _PLD object. In addition, two fields in the _PLD object: Group token (Bit 86:79) and Group Position (Bit 94:87) should be correctly defined for all USB

connection points, including those that are not visible externally (which should be indicated by setting bit 64 to zero).

No two USB connection points should have identical combination of Group token and Group Position. If two ports are sharing a connection point, they should have identical _PLD objects.

This information helps define the mapping of USB ports to uniquely identifiable connection points. The Windows USB 3.x stack will use this information to determine which ports are tied to the same connection points. Any USB port that does not have a _PLD object will be assumed to be not connectable and not visible (i.e. it is not being used at all). The definition of connectable port as per ACPI 4.0 spec (section 9.13), is a port on which either a user can connect a device OR there is an integrated device connected to it.

Please see design notes for additional information on how to implement this requirement.

Design Notes:

Example

This example is based on xHCI Spec (Version .95) Appendix D. The hardware configuration is exactly the same as in that Appendix. The ACPI representation of that hardware configuration differs in this example; those differences are highlighted.

The following is an example of the ACPI objects defined for an xHCI that implements a High-speed and SuperSpeed Bus Instance that are associated with USB2 and USB3 Protocol Root Hub Ports, respectively. The xHCI also supports an integrated High-speed hub to provide Low- and Full-speed functionality. The External Ports defined by the xHC implementation provide either a USB2 data bus (i.e. a D+/D- signal pair) or a SuperSpeed (or future USB speed) data bus (i.e. SSRx+/SSRx- and SSTx+/SSTx-signal pairs).

Where:

- The motherboard presents 5 user visible connectors C1 - C5:
  - Motherboard connectors C1 and C2 support USB2 (LS/FS/HS) devices.
  - Motherboard connectors C3, C4 and C5 support USB3 (LS/FS/HS/SS) devices.
- The xHCI implements a High-speed Bus Instance associated with USB2 Protocol Root Hub ports, e.g. HCP1 and HCP2 are High-speed only, i.e. they provide no Low- or Full-speed support.
- The xHCI presents 7 External Ports (P1 - P7):
  - External Port 1 (P1) is HS only and is not visible or connectable.
  - External Ports 2 - 5 (P2 - P5) support LS/FS/HS devices:
    - P2 is attached to motherboard USB2 connector C1.
    - P3 is attached to motherboard USB2 connector C2.

        - P4 is attached to the USB 2.0 logical hub of the Embedded USB3 Hub on the motherboard. The USB 2.0 logical hub supports the LS/FS/HS connections for 2 ports (EP1 - EP2)
        - The USB 2.0 connections of motherboard hub ports EP1 and EP2 are attached to motherboard connectors C3 and C4 respectively, providing the LS/FS/HS support for the USB3 connectors.
    - P5 is attached to motherboard connector C5, providing the LS/FS/HS support to the motherboard USB3 connector C5.
    - External Port 6 (P6) is attached to the SuperSpeed logical hub of the Embedded USB3 Hub on the motherboard. The SuperSpeed logical hub supports the SS connections of 2 ports (EP1 - EP2).
    - The SuperSpeed connections of motherboard hub ports EP1 and EP2 are attached to motherboard connectors C3 and C4 respectively, providing the SS support for the USB3 connectors.
    - External Port 7 (P7) is attached to motherboard connectors C5, providing the SS support for the USB3 connector.
- The xHCI implements 4 internal HS Root Hub ports (HCP1 - HCP4), 2 High-speed and 2 SuperSpeed:
    - Internal Port 1 (HCP1) maps directly to External Port 1 (P1).
    - Internal Port 2 (HCP2) is attached to a HS Integrated Hub. The Integrated Hub supports 4 ports (IP1 - IP4):
        - Ports 1 to 4 (IP1-IP4) of the Integrated Hub attach to External Ports 2 to 5 (P2-P5), respectively.
    - Internal Ports 3 and 4 (HCP3, HCP4) attach to External Ports 6 and 7 (P6, P7), respectively.
- All connectors are located on the back panel and assigned to the same Group.
- Connectors C1 and C2 are USB2 compatible and their color is not specified. Connectors C3 to C5 are USB3 compatible and their color is specified.
- External Ports P1 - P5 present a USB2 data bus (i.e. a D+/D- signal pair). External Ports P6 and P7 present a SuperSpeed data bus (i.e. SSRx+/SSRx- and SSTx+/SSTx- signal pairs).

```
Scope( \_SB ) {
        Device( PCI0 ) {
// Host controller ( xHCI )
Device( USB0 ) {
// PCI device#/Function# for this HC. Encoded as specified in the ACPI
// specification
Name( _ADR, 0xyyyyzzzz )
// Root hub device for this HC #1.
Device( RHUB ) {
Name( _ADR, 0x00000000 ) // must be zero for USB root hub
// Root Hub port 1 ( HCP1 )
Device( HCP1 ) {// USB0.RHUB.HCP1
Name( _ADR, 0x00000001 )
// USB port configuration object. This object returns the system
// specific USB port configuration information for port number 1
Name( _UPC, Package() {
0x01,// Port is connectable but not visible
0xFF,// Connector type (N/A for non-visible ports)
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
} // Device( HCP1 )
// Root Hub port 2 ( HCP2 )
Device( HCP2 ) {// USB0.RHUB.HCP2
Name( _ADR, 0x00000002 )
Name( _UPC, Package() {
0xFF,// Port is connectable
0x00,// Connector type - (N/A for non-visible ports)
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// Even an internal connection point should have a _PLD and
// provide a valid Group Token and Position
Name( _PLD, Buffer( 0x10) {
0x00000081,// Revision 1,
// color width height ignored for non-visible connector
```

```
0x00000000,// connector type ignored for non-visible connector
0x00808000,// Not User visible, Panel, position shape ignored,
// Group Token = 1, Group Position = 1
// This is the group of all internal connectors.
// Each connector should have a unique position in this
// group
0x00000000} )// Ignored for non-visible connectors
//
// There is no separate node for the integrated hub itself
//
// Integrated hub port 1 ( IP1 )
Device( IP1 ) {// USB0.RHUB.HCP2.IP1
// Address object for the port. Because the port is
// implemented on integrated hub port #1, this value must be 1
Name( _ADR, 0x00000001 )
Name( _UPC, Package() {
0xFF,// Port is connectable
0x00,// Connector type - Type 'A'
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// provide physical connector location info
Name( _PLD, Buffer( 0x10) {
0x00000081,// Revision 1, Ignore color
// Color (ignored), width and height not
0x00000000,// required as this is a standard USB 'A' type
// connector
0x00800c69,// User visible, Back panel, Center, left,
// shape = vert. rect, Group Token = 0,
// Group Position 1 (i.e. Connector C1)
0x00000003} )// ejectable, requires OPSM eject assistance
} // Device( IP1 )
// Integrated Hub port 2 ( IP2 )
Device( IP2 ) {// USB0.RHUB.HCP2.IP2
// Address object for the port. Because the port is
```

```
// implemented on integrated hub port #2, this value must be 2
Name( _ADR, 0x00000002 )
Name( _UPC, Package() {
0xFF,// Port is connectable
0x00,// Connector type - Type 'A'
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// provide physical connector location info
Name( _PLD, Buffer( 0x10) {
0x00000081,// Revision 1, Ignore color
// Color (ignored), width and height not
0x00000000,// required as this is a standard USB 'A' type
// connector
0x01000c69,// User visible, Back panel, Center, Left,
// Shape = vert. rect, Group Token = 0,
// Group Position 2 (i.e. Connector C2)
0x00000003} )// ejectable, requires OPSM eject assistance
} // Device( IP2 )
// Integrated Hub port 3 ( IP3 )
Device( IP3 ) { // USB0.RHUB.HCP2.IP3
// Address object for the port. Because the port is implemented
// on integrated hub port #3, this value must be 3
Name( _ADR, 0x00000003 )
// Must match the _UPC declaration for USB0.RHUB.HCP3 as
// this port shares the connection point
Name( _UPC, Package() {
0xFF,// Port is not connectable
0x00,// Connector type - (N/A for non-visible ports)
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// Even an internal connection point should have a _PLD and
// provide a valid Group Token and Position.
// Must match the _PLD declaration for USB0.RHUB.HCP3 as
// this port shares the connection point
```

```
Name( _PLD, Buffer( 0x10) {
0x00000081,// Revision 1,
// color width height ignored for non-visible connector
0x00000000,// connector type ignored for non-visible connector
0x01008000,// Not User visible, Panel, position shape ignored,
// Group Token = 1, Group Position = 2
// This is the group of all internal connectors.
// Each connector should have a unique position in this
// group
0x00000000} )// Ignored for non-visible connectors
//
// There is no separate node for the embedded hub itself
//

// Motherboard Embedded Hub 2.0 Logical Hub port 1 ( EP1 )
Device( EP1 ) {// USB0.RHUB.HCP2.IP3.EP1
Name( _ADR, 0x00000001 )
// Must match the _UPC declaration for
// USB0.RHUB.HCP3.EP1 as this port provides
// the LS/FS/HS connection for C3
Name( _UPC, Package() {
0xFF,// Port is connectable
0x03,// Connector type - USB 3 Type 'A'
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// provide physical connector location info
Name( _PLD, Buffer( 0x10) {
0x0072C601,// Revision 1, Color valid
// Color (0072C6h), width and height not
0x00000000,// required as this is a standard USB
// 'A' type connector
0x01800c69,// User visible, Back panel, Center,
// Left, shape = vert.
// rect, Group Token = 0,
```

```
// Group Position 3
//(i.e. Connector C3)
0x00000003} )// ejectable, requires OPSM eject
// assistance
} // Device(EP1)
// Motherboard Embedded Hub 2.0 Logical Hub port 2 ( EP2 )
Device( EP2 ) {// USB0.RHUB.HCP2.IP3.EP2
Name( _ADR, 0x00000002 )
// Must match the _UPC declaration for
// USB0.RHUB.HCP3.EHUB.EP2 as this port provides
// the LS/FS/HS connection for C4
Name( _UPC, Package() {
0xFF,// Port is connectable
0x03,// Connector type - USB 3 Type 'A'
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// provide physical connector location info
Name( _PLD, Buffer( 0x10) {
0x0072C601,// Revision 1, Color valid
// Color (0072C6h), width and height not
0x00000000,// required as this is a standard USB
// 'A' type connector
0x02000c69,// User visible, Back panel, Center,
// Left, Shape = vert.
// rect, Group Token = 0,
// Group Position 4 (i.e. Connector C4)
0x00000003} )// ejectable, requires OPSM eject
//assistance
} // Device( EP2 )
} // Device( IP3 )
// Integrated hub port 4 ( IP4 )
Device( IP4 ) { // USB0.RHUB.HCP2.IP4
Name(_ADR, 0x00000004)
// Must match the _UPC declaration for USB0.RHUB.HCP4 as
```

// this port provides the LS/FS/HS connection for C5

Name(_UPC, Package() {

0xFF,// Port is connectable

0x03,// Connector type - USB 3 Type 'A'

0x00000000,// Reserved 0 - must be zero

0x00000000})// Reserved 1 - must be zero

// provide physical connector location info

Name(_PLD, Buffer(0x10) {

0x0072C601,// Revision 1, Color valid

// Color (0072C6h), width and height not

0x00000000,// required as this is a standard USB 'A' type

// connector

0x02800c69,// User visible, Back panel, Center, Left,

// Shape = vert. rectangle, Group Token = 0,

// Group Position 5 (i.e. Connector C5)

0x00000003})// ejectable, requires OPSM eject assistance

} // Device(IP4)

} // Device(HCP2)

// Root Hub port 3 (HCP3)

Device(HCP3) {

Name(_ADR, 0x00000003)

// Must match the _UPC declaration for USB0.RHUB.HCP2.IP3 **as**

// this port shares the connection point

Name(_UPC, Package() {

0xFF,// Port is connectable

0x00,// Connector type - (N/A for non-visible ports)

0x00000000,// Reserved 0 - must be zero

0x00000000})// Reserved 1 - must be zero

// Even an internal connection point should have a _PLD and

// provide a valid Group Token and Position.

// Must match the _PLD declaration for USB0.RHUB.HCP2.IP3 **as**

// this port shares the connection point

Name(_PLD, Buffer(0x10) {

0x00000081,// Revision 1,

```
// color width height ignored for non-visible connector
0x00000000,// connector type ignored for non-visible connector
0x01008000,// Not User visible, Panel, position shape ignored,
// Group Token = 1, Group Position = 2
// This is the group of all internal connectors.
// Each connector should have a unique position in this
// group
0x00000000} )// Ignored for non-visible connectors
//
// There is no separate node for the embedded hub itself
//
// Motherboard Embedded Hub SS Logical Hub port 1 ( EP1 )
Device( EP1 ) {// USB0.RHUB.HCP3.EP1
Name( _ADR, 0x00000001 )
// Must match the _UPC declaration for
// USB0.RHUB.HCP2.IHUB.IP3.EHUB.EP1 as this port
// provides the SS connection for C3
Name( _UPC, Package() {
0xFF,// Port is connectable
0x03,// Connector type - USB 3 Type 'A'
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// provide physical connector location info
Name( _PLD, Buffer( 0x10) {
0x0072C601,// Revision 1, Color valid
// Color (0072C6h), width and height not
0x00000000,// required as this is a standard USB
// 'A' type connector
0x01800c69,// User visible, Back panel, Center,
// Left, shape = vert.
// rect, Group Token = 0,
// Group Position 3
//(i.e. Connector C3)
0x00000003} )// ejectable, requires OPSM eject
```

```
// assistance
} // Device(EP1)
// Motherboard Embedded Hub SS Logical Hub port 2 ( EP2 )
Device( EP2 ) {// USB0.RHUB.HCP2.EP2
Name( _ADR, 0x00000002 )
// Must match the _UPC declaration for
// USB0.RHUB.HCP3.IP3.EP2 as this port
// provides the SS connection for C4
Name( _UPC, Package() {
0xFF,// Port is connectable
0x03,// Connector type - USB 3 Type 'A'
0x00000000,// Reserved 0 - must be zero
0x00000000} )// Reserved 1 - must be zero
// provide physical connector location info
Name( _PLD, Buffer( 0x10) {
0x0072C601,// Revision 1, Color valid
// Color (0072C6h), width and height not
0x00000000,// required as this is a standard USB
// 'A' type connector
0x02000c69,// User visible, Back panel, Center,
// Left, Shape = vert.
// rect, Group Token = 0,
// Group Position 4 (i.e. Connector C4)
0x00000003} )// ejectable, requires OPSM eject
// assistance
} // Device( EP2 )
} // Device( HCP3 )
// Root Hub port 4 ( HCP4 )
Device( HCP4 ) {// USB0.RHUB.HCP4
Name( _ADR, 0x00000004 )
// Must match the _UPC declaration for USB0.RHUB.HCP2.IP4 as
// this port provides the SS connection for C5
Name( _UPC, Package() {
0xFF,// Port is connectable
```

0x03,// Connector type - USB 3 Type 'A'

0x00000000,// Reserved 0 - must be zero

0x00000000})// Reserved 1 - must be zero

// provide physical connector location info

Name(_PLD, Buffer(0x10) {

0x0072C601,// Revision 1, Color valid

// Color (0072C6h), width and height not

0x00000000,// required as this is a standard USB 'A' type

// connector

0x02800c69,// User visible, Back panel, Center, Left,

// Shape = vert. rect, Group Token = 0,

// Group Position 5 (i.e. Connector C5)

0x00000003})// ejectable, requires OPSM eject assistance

} // Device(HCP4)

} // Device(RHUB)

} // Device(USB0)

//

// Define other control methods, etc.

} // Device(PCIO)

} // Scope(_SB)

System.Fundamentals.SystemUSB.XhciSupportsMinimum31Streams

xHCI controller must support at least 31 primary streams per endpoint.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Refer to the eXtensible Host Controller Interface specification, section 4.12.2.

This requirement is for the MaxPSASize in the HCCPARAMS to be set to 4 at the minimum to enable ultimate data transfer rate with UAS devices.

Storage devices based on the USB Attached SCSI Protocol (UASP) will utilize streams to achieve faster data transfer rates. To enable the best experience with these devices, every xHCI controller will need to support at least 31 primary streams.

Send comments about this topic to Microsoft

System.Fundamentals.TPM20

In this topic:

System.Fundamentals.TPM20.EKCerts

Systems shipping with TPM 2.0 must contain a full endorsement key certificate.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

All TPMs must contain a full Endorsement Key (EK) certificate stored in the TPM NV RAM as described in the TCG PC Client Specific Implementation Specification for Conventional BIOS, section 7.4.5: TCG_FULL_CERT, or be capable of being retrieved by the device during the first boot experience.. The NV RAM index used for storing the EK certificate must be pre-defined and created with the value of 0x01c00002. If a nonce or template is used to create an EK Certificate which is different from the defaults used by Windows, the same must be specified in the TPM NVRAM under the index of 0x01c00003 for the EK nonce and under the index of 0x01c00004 for the template.

TPM Artifact	NV Index	Required
EK Certificate	0x01c00002	Yes
EK nonce	0x01c00003	Optional, required only if non-default values are used
EK template	0x01c00004	Optional, required only if non-default values are used

The EK certificate must be capable of written to the TPM NVRAM in such a way that the action of clearing the TPM does not delete the EK certificate.

The certificate must have the EKU specified that indicates that it is indeed an EK Certificate. The OID used for this purpose should be "2.23.133.8.1".

The EK certificate could also be signed using ECDSA. The supported ECC curves for this purpose are NIST 256, 384 and 521. Note that the Endorsement Key is still a RSA 2048 bit key.

The EK certificate must contain an AIA extension that contains the URL for the issuing CA Certificate in the certificate chain. AIA extension (Authority information access locations) must also be present in each non-root cert in the chain with URLs that make the issuing CA certificate (any intermediate CA certs or the root CA cert) – all discoverable and retrievable when starting only with a single EK cert. For more information on AIA extension, please refer to http://technet.microsoft.com/en-us/library/cc753754.aspx.

Note: The EK certificate may be created by the TPM manufacturer or the Platform manufacturer.

System.Fundamentals.TPM20.TPM20

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

A system that implements a Trusted Platform Module must include a TPM which complies with TCG Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 0.99 (or later), 1.16 or later is recommended.

Mandatory, All systems must contain a TPM 2.0 which complies with the requirements in this section. For the specified SKUs above, the TPM must be both available to the system and enabled in the shipping configuration, except as noted below.

- This requirement is **Optional** on Client x86 and x64 SKUs for 365 days after RTM, then it becomes **Mandatory**.
- For all Mobile SKUs this is **Mandatory** at RTM.
- For Client SKU Devices which ship with silicon that supports a firmware TPM (fTPM), or which contain a dTPM, this requirement is **Mandatory** at RTM regardless of the above 365 day window.
- This requirement is **Optional** for Server x64.

Dual TPM Systems: A Device may optionally include a TPM 1.2 In such a case, the Device Manufacturer may choose to enable the TPM 1.2 by default. If the System also contains a TPM 2.0, it must be possible to toggle between TPMs if more than one is available. This statement does not supersede any requirement for the inclusion of a TPM 2.0.

Systems with TPM 2.0 must comply with the following requirements:

- The platform shall implement all "Required" features in the table below. Support for "Recommended" entries is strongly encouraged.

Integrity Feature	SOC Hardware Functionality	Required	Recommended
Trusted Execution Environment	**Isolated Storage** Availability of storage for storing long term secrets. This storage must not be possible to modify by the OS without detection by pre-Operating System components.	X	
	Secure (isolated from runtime OS) storage of: • Values (such as an endorsement primary seed) that survive complete platform power off as well as firmware updates • Values (such as a NV counters) that survive complete platform power off but do not necessarily survive firmware updates (in this case these values shall be reset to a random value) • Values (such as the Platform Configuration Registers) that survive platform power-down to the equivalent of ACPI S3 if TPM2_Shutdown(TPM_SU_STATE) is called but may be lost on further power-down.	X	
Platform Attestation	Boot measurements recorded in the Platform Configuration Registers for all firmware code loaded after the establishment of the Core Root of Trust for Measurement.	X	
	Implementation of PCRs 0 through 23 for SHA-256, dedicated to the same boot measurements as TPM 1.2.	X	
	Support for SHA-1, SHA-256, AES-128, and RSA-2048 algorithms.	X	

	Robustness against side channel attacks including Differential Power Analysis (DPA) and Electromagnetic Emanations (EM)		X
Commands	The TPM2_HMAC command is required.	X	
	Support of ECC algorithms in TPM 2.0 (Note: the TPM library specification requires the support of TPM_ALG_ECDSA and TPM_ALG_ECDH if a TPM implements the TPM_ALG_ECC curve hence are part of requirements) TPM_ECC_NIST_P256 curve should be supported, as specified in Table 8 of TPM library specification Part 2		X
	If ECC algorithms are supported the following commands are requisite to enable use of ECC. In TPM 2.0. These were "recommended" in Windows 8 requirements (details can be found in TCG TPM library specification Part 3) • TPM2_ECDH_KeyGen • TPM2_ECDH_ZGen • TPM2_ECC_Parameters • TPM2_Commit		X
Physical Presence Interface	Mobile SKUs: NoPPIClear must be set to TRUE. Server SKUs: NoPPIClear value may be chosen by Device Manufacturer. Client SKUs: NoPPIClear must be set to FALSE.	X	

- A TPM that does not support a separate cryptographic processing unit isolated from the main CPU(s) must support a Trusted Execution Mode. The Trusted Execution Mode must have a higher privilege level than the Normal Execution Mode, giving it access to data and code not available to the Normal Execution Mode.
- During the boot sequence, the boot firmware/software shall measure all firmware and all software components it loads after the core root of trust for measurement is established. The measurements shall be logged and extended to platform configuration registers in a manner compliant with the following requirements.

- The measurements must be implemented such a third party can reliably and verifiably identify all components in the boot process until the boot completes successfully or a compromised component was loaded.
    - For example, if the third component loaded includes an exploited vulnerability, then values for the first, second, and third component in the trusted boot log correctly reflect the software that loaded.
    - To achieve this, the trusted execution environment must provide a mechanism of signing the values of the registers used for Trusted Boot. The interface to the signature ("attestation") mechanism shall comply with the requirements defined in Microsoft Corporation, "Trusted Execution Environment ACPI Profile, 1.0 dated March 2, 2012".
- The system shall include a trusted execution environment supporting the command set defined in Microsoft Corporation, "TPM v2.0 Command and Signature Profile, 1.0 dated March 2, 2012."
- The system shall support the interface specified in Microsoft Corporation, "Trusted Execution Environment ACPI Profile, 1.0 dated March 2, 2012".
- The system shall support the interface and protocol specified in Microsoft Corporation, "Trusted Execution Environment EFI Protocol, 1.0 dated March 2, 21012".
- The system is required to measure data into PCR [7] as specified in TCG EFI Platform Specification For TPM Family 1.1 or 1.2 Specification Version 1.22 Revision 15 27 January 2014. The UEFI firmware update process must also protect against rolling back to insecure firmware versions, or non-production versions that may disable secure boot or include non-production keys. A physically present user may however override the rollback protection manually. In such a scenario (where the rollback protection is overridden), the TPM must be cleared.
- Platform firmware must ensure invariance of PCRs 0, 2, and 4 across power cycles in the absence of changes to the platform's static core root of trust for measurements (SRTM). Platform firmware must ensure invariance of PCR[7] if implemented as specified in Microsoft Corporation, "Trusted Execution Environment EFI Protocol,.1.0 dated March 2, 2012" across power cycles in the absence of changes to the platform's static core SRTM. Attaching a (non-bootable) USB to the platform or attaching the platform to a docking station shall not cause changes to the SRTM.

- Execution of the TPM 2.0 command TPM2_NV_Increment must not require an open object slot.
- TPM must meet performance requirements as stated below:

  Prior to exit boot services, TPM shall complete extend operations within 20msec. After exit boot services, TPM Extend operations shall complete within 20mS when a TPM is not in a lower power state. If a TPM is in a low power state Extend operations shall complete in 20mS after the TPM has exited its low power state.

Mandatory: All client SKUs are required to ship TPM 2.0 with SHA2 PCR banks. (Note: It is acceptable to ship TPMs with a single switchable PCR bank that can be utilized for both SHA 1 and SHA 2 measurements.)

Mandatory: SHA-2 Event logs must be supported.

Send comments about this topic to Microsoft

System.Fundamentals.TrustedPlatformModule

A Trusted Platform Module (TPM) is a microchip designed to provide basic security related functions. Requirements in this area reflect the required TPM version and compatibility with Windows Bitlocker.

In this topic:

System.Fundamentals.TrustedPlatformModule.TPMCompliesswithTCGTPMMainSpecification

A system that implements a Trusted Platform Module (TPM) 1.2 must include a TPM that complies with the TCG TPM Main Specification, Version 1.2, Revision 103 (or a later revision), parts 1, 2 and 3.

Applies to	Windows 10 x64 Windows 10 x86

Description

A system that implements a Trusted Platform Module (TPM) 1.2 must include a TPM that complies with the TCG TPM Main Specification, Version 1.2, Revision 103 (or a later revision), parts 1, 2 and 3.

(http://www.trustedcomputinggroup.org/resources/tpm_main_specification)

The TPM must meet the following additional requirements:

- The time required for the TPM to perform all the self-testing performed by the TPM_ContinueSelfTest command must be less than 1 second.
- If the TPM receives a command, after receipt of the TPM_ContinueSelfTest command, but prior to the completion of the TPM_ContinueSelfTest self-test actions, it must not return TPM_NEEDS_SELFTEST.
- The TPM's monotonic counter must be designed to increment at least twice per a platform boot cycle.
- The TPM must implement the TPM_CAP_DA_LOGIC capability for the TPM_GetCapability command.
- By default, the TPM dictionary attack logic must permit at least 9 authorization failures in an a 24 hour time period before entering the first level of defense. Small durations of lockout for less than five seconds are acceptable within a 24 hour period with 9 authorization failures if the TPM leaves the lockout state automatically after five seconds elapses. Alternately, the default system image must contain non-default software anti-hammering settings which correspond to TPM default behavior. (In the Windows 8 OS the settings can be seen by running gpedit.msc then expanding the following in the left tree view: Local Computer Policy\Computer Configuration\Administrative Templates\System\Trusted Platform Module Services. The values to customize are: Standard User Lockout Duration, Standard User Individual Lockout Threshold, and Standard User Total Lockout Threshold.)
- The TPM dictionary attack logic must not permit more than 5000 authorization failures per a year.
- To help platform manufacturers achieve as fast of a boot as possible, the shorter the TPM_ContinueSelfTest execution time, the better.
- It is recommended platform manufacturers provide information about the TPM's dictionary attack logic behavior in customer documentation that includes explicit steps to recover after the TPM enters a locked out state.
- It is recommended the TPM state when shipped is enabled and activated.

Note: Windows uses more TPM functionality than previous releases so Windows Certification Tests for the TPM are more extensive.

System.Fundamentals.TrustedPlatformModule.TPMEnablesFullUseThroughSystemFirmware

Systems with Trusted Platform Modules enable full use of the TPM including system firmware enhancements.

Applies to	Windows 10 x64 Windows 10 x86

Description

A system that implements a Trusted Platform Module 1.2 (TPM) must support specific system firmware enhancements. The system firmware code must participate in a measured chain of trust that is established for the pre-operating system boot environment at each power cycle. The system firmware code must support the protected capabilities of the TPM v1.2 and must maintain the SRTM chain of trust.

For a system that implements a Trusted Platform Module (TPM) 1.2, the platform must comply with the following specifications:

- The TCG Platform Reset Attack Mitigation Specification Version 1.00, Revision .92 or later. Revision 1.00 is strong encouraged, including implementation of detecting an orderly OS shutdown(http://www.trustedcomputinggroup.org/resources/pc_client_work_group_platform_reset_attack_mitigation_specification_version_10).
- The TCG Physical Presence Interface Specification Version 1.0 Revision 1.00 or later (http://www.trustedcomputinggroup.org/resources/tcg_physical_presence_interface_specification).
- The TCG PC Client Work Group PC Client Specific TPM Interface Specification (TIS) Version 1.20 Revision 1.00 or later. Implementing version 1.21 or later is strongly encouraged. The TPM must implement the memory mapped space described in the specification. (http://www.trustedcomputinggroup.org/resources/pc_client_work_group_pc_client_specific_tpm_interface_specification_tis)
- If the platform implements UEFI firmware, it must implement:
  - The TCG EFI Platform Specification Version 1.20, Revision 1.00 or later, (http://www.trustedcomputinggroup.org/resources/tcg_efi_platform_specification_version_120_revision_10).

- The TCG EFI Protocol Version 1.20, Revision 1.00 or later, http://www.trustedcomputinggroup.org/resources/tcg_efi_protocol_version_120_revision_10.

- If the platform implements conventional BIOS, it must implement the PC Client Workgroup Specific Implementation Specification for Conventional BIOS, Version 1.20, Revision 1.00 or later. Note: Implementing the errata version 1.21 is strongly recommended. (http://www.trustedcomputinggroup.org/resources/pc_client_work_group_specific_implementation_specification_for_conventional_bios_specification_version_12)

- The TCG ACPI General Specification Version 1.00, Revision 1.00 (http://www.trustedcomputinggroup.org/resources/server_work_group_acpi_general_specification_version_10).

- The Windows Vista BitLocker Client Platform Requirements, dated May 16, 2006, or later, available at http://go.microsoft.com/fwlink/?LinkId=70763. For conventional BIOS systems instead of the PCR measurements listed in the Windows Vista BitLocker Client Platform Requirements, a system may implement the PCR measurements defined in the TCG PC Client Workgroup Specific Implementation Specification for Conventional BIOS, Version 1.21, Revision 1.00.

Design Notes:

The TPM provides a hardware root of trust for platform integrity measurement and reporting. The TPM also provides operating system independent protection of sensitive information and encryption keys.

System.Fundamentals.TrustedPlatformModule.TPMRequirements

System implementing TPM 1.2 must meet requirements.

Applies to	Windows 10 x64 Windows 10 x86

Description

For a system that implements a Trusted Platform Module (TPM) 1.2, the platform must comply with the following specifications:

- The PC Client Work Group Platform Reset Attack Mitigation Specification, Version 1.0 Version 1.00, Revision 1.00 or later, http://www.trustedcomputinggroup.org/resources/pc_client_work_group_platform_reset_attack_mitigation_specification_version_10.

- The TCG Physical Presence Interface Specification Version 1.2, Revision 1.00, http://www.trustedcomputinggroup.org/resources/tcg_physical_presence_interface_specification.
- The TCG PC Client Work Group PC Client Specific TPM Interface Specification (TIS) Version 1.21 Revision 1.00 or later (http://www.trustedcomputinggroup.org/resources/pc_client_work_group_pc_client_specific_tpm_interface_specification_tis).
- If the platform implements UEFI firmware, it must implement
  - The TCG EFI Platform Specification Version 1.20, Revision 1.0 or later, http://www.trustedcomputinggroup.org/resources/tcg_efi_platform_specification_version_120_revision_10.
  - The TCG EFI Protocol Version 1.20, Revision 1.00 or later, http://www.trustedcomputinggroup.org/resources/tcg_efi_protocol_version_120_revision_10.
- If the platform implements conventional BIOS, it must implement the PC Client Workgroup Specific Implementation Specification for Conventional BIOS, Version 1.20, Revision 1.00 or later. Note: The errata version 1.21 is strongly recommended instead. (http://www.trustedcomputinggroup.org/resources/pc_client_work_group_specific_implementation_specification_for_conventional_bios_specification_version_12).
- The TCG ACPI General Specification Version 1.00, Revision 1.00, http://www.trustedcomputinggroup.org/resources/server_work_group_acpi_general_specification_version_10.
- Windows Vista BitLocker Client Platform Requirements, dated May 16, 2006, or later (http://go.microsoft.com/fwlink/p/?LinkId=70763). For conventional BIOS systems instead of the PCR measurements listed in the Windows Vista BitLocker Client Platform Requirements, a system may implement the PCR measurements defined in the TCG PC Client Workgroup Specific Implementation Specification for Conventional BIOS, Version 1.21, Revision 1.00.

And the system MUST meet the following additional requirements:

- Platform firmware must ensure invariance of PCRs 0, 2 and 4 and also PCR 7 if implemented as specified in Appendix A of the Microsoft Corporation, "Trusted Execution Environment EFI Protocol, 1.00 dated March 2nd, 2012" document across power cycles in the absence of changes to the platform's static root of trust for measurements (SRTM). Attaching a (non-

bootable) USB to the platform or attaching the platform to a docking station shall not cause changes to the SRTM.

- If the platform is a server platform, it must reserve PCRs 8 through 15 for OS use. (Note: The same requirement is true for client platforms.)
- The platform must implement the memory mapped space for the TPM interface (for example, legacy port based I/O is not sufficient).
- The system firmware must perform Physical Presence Interface operations when the platform is restarted. It is strongly recommended the system firmware performs Physical Presence Interface operations also after shutdown. (Specifically, the Physical Presence Interface Specification Version 1.2, section 2.1.4: Get Platform-Specific Action to Transition to Pre-OS Environment must return a value of 2: Reboot.) (This requirement allows remote administrators to perform Physical Presence Operations without needing to be physically present to turn the platform back on.)
- The default configuration for the system must have the NoPPIProvision flag specified in the TCG Physical Presence Interface Specification, section 2: Physical Presence Interface set to TRUE.
- The default system firmware configuration must allow the OS to request Physical Presence operations 6, 7, 10, and 15. Note: The operations are described in Table 2 of the TCG Physical Presence Interface Specification Version 1.2, Revision 1.00.
- If the system implements the NoPPIClear flag it should do so as specified in the TCG Physical Presence Interface Specification, section 2: Physical Presence Interface. The platform should either provide a system firmware configuration setting to change the flag or implement physical presence operations 17 and 18. Note: The operations and the NoPPIClear flag are described in Table 2 of the TCG Physical Presence Interface Specification Version 1.2, Revision 1.00. (Implementing this flag helps facilitate automated testing of the physical presence interface during Windows certification testing and permits managed environments to completely automate TPM management from the OS without physical presence if an enterprise decides to set the NoPPIClear flag.)
- The system firmware must implement the _DSM Query method (function index 0) in addition to the Physical Presence Interface methods defined in the TCG Physical Presence Interface Specification, section 2: ACPI Functions. (Please refer to the Advanced Configuration and

Power Interface Specification Revision 5.0 for an implementation example of the _DSM Query method.)

- The system firmware must implement the _DSM Query method (function index 0) in addition to the Memory Clear Interface method defined in the TCG Platform Reset Attack Mitigation Specification, section 6: ACPI _DSM Function. (Please refer to the Advanced Configuration and Power Interface Specification Revision 5.0 for an implementation example of the _DSM Query method.)
- The system firmware must implement the auto detection of clean OS shutdown and clear the memory overwrite bit as defined in the TCG Platform Reset Attack Mitigation Specification, section 2.3: Auto Detection of Clean Static RTM OS Shutdown. Exception: If the system is able to unconditionally clear memory during boot without increasing boot time, the system may not implement the auto detection (however the pre-boot and ACPI interface implementations are still required).
- When the system is delivered to an end customer, the TPM permanent flag TPM_PF_NV_LOCKED must be set to TRUE. (This requirement is for systems. For motherboards the flag may be set to FALSE when delivered to the platform manufacturer, however instructions/tools must advise platform manufacturers to set the flag to TRUE before delivery to end customers.)
- When the system is delivered to an end customer, the TPM permanent flag TPM_PF_NV_ PHYSICALPRESENCELIFETIMELOCK must be set to TRUE. (This requirement is for systems. For motherboards the flag may be set to FALSE when delivered to the platform manufacturer, however instructions/tools must advise platform manufacturers to set the flag to TRUE before delivery to end customers.)
- The system must contain a full Endorsement Key (EK) certificate stored in the TPM NV RAM as described in the TCG PC Client Specific Implementation Specification for Conventional BIOS, section 7.4.5: TCG_FULL_CERT. The NV RAM index used for storing the EK certificate must be the pre-defined and reserved index TPM_NV_INDEX_EKCert as defined the TPM Main Specification, Part 2, section 19.1.2: Reserved Index Values. As recommended in the TCG PC Client Specific Implementation Specification for Conventional BIOS, section 4.2.1: TPM Main Specification Reserved Indexes, the D bit attribute must be set for the index. (Note: The certificate may be created by the TPM manufacturer or the platform manufacturer.) Exception: If the system supports generation of a new EK it is not required (but is still strongly recommended) to have an EK certificate.

- The system firmware must ship with the TPM enumerated by default. (This means the ACPI device object for the TPM must be present by default in the system ACPI tables.)
- The system firmware must support clearing the TPM from within a setup menu.
- At least 256 bytes of TPM NVRAM must be reserved (and available) for OS use.
- The firmware must issue the TPM_ContinueSelfTest during boot such that the self-test completes before the OS loader is launched.
    - If the TPM device performs the self-test synchronously, the firmware TPM driver should be optimized to issue the command to the device but allow the boot process to proceed without waiting for the return result from the TPM_ContinueSelfTest command. If the firmware TPM driver receives a subsequent command, it should delay the subsequent command until the TPM_ContinueSelfTest command completes instead of aborting the TPM_ContinueSelfTest command.)
    - A recommendation is to start the self-test before some action which takes at least one second but does not have a dependency on the TPM.
- The platform may or may not issue the TPM_ContinueSelfTest upon resume from S3 (sleep).
- The ACPI namespace location for the TPM device object must only depend on the System Bus, ISA or PCI bus drivers provided by Microsoft. The System Bus does not have an ID, but is identified as _SB. The ISA bus device IDs may be PNP0A00 or PCI\CC_0601. The PCI bus device IDs may be PNP0A03 or PCI\CC_0604. In addition, the TPM device object may also depend on these generic bridges, containers or modules: PNP0A05, PNP0A06 and ACPI0004. No other device dependencies are permitted for the ACPI namespace location for the TPM device object.
- Optional. The platform is recommended to support measurements into PCR [7] as specified in Appendix A of Microsoft Corporation, "Trusted_Execution_Environment_EFI_Protocol, 1.00 dated March 2, 2012" specification, The UEFI firmware update process must also protect against rolling back to insecure firmware versions, or non-production versions that may disable secure boot or include non-production keys. A physically present user may however override the rollback protection manually. In such a scenario (where the rollback protection is overridden), the TPM must be cleared.

Note: Bitlocker utilizes PCR7 for better user experience and limit PCR brittleness. If Secure Boot launch of Windows BootMgr requires use of an Allowed DB entry other than the Microsoft-provided EFI_CERT_X509 signature with "CN=Microsoft Windows Production PCA 2011" and "Cert Hash(sha1):

58 0a 6f 4c c4 e4 b6 69 b9 eb dc 1b 2b 3e 08 7b 80 d0 67 8d, Bitlocker will then not be able to utilize PCR7. It is recommended that the only Allowed DB entry for Secure Boot are Microsoft-provided EFI_CERT_X509 signature with "CN=Microsoft Windows Production PCA 2011" and "Cert Hash(sha1): 58 0a 6f 4c c4 e4 b6 69 b9 eb dc 1b 2b 3e 08 7b 80 d0 67 8d.

"Trusted Execution Environment EFI Protocol, 1.00 dated March 2, 2012" specification must be requested explicitly from Microsoft. To acquire the current version, first check for its availability on the Microsoft Connect site. If it is not available, contact http://go.microsoft.com/fwlink/?LinkId=237130.

Send comments about this topic to Microsoft

System.Fundamentals.USBBoot

The feature and requirements are about being able to boot from a USB device.

In this topic:

System.Fundamentals.USBBoot.BootFromUSB

System firmware supports booting from all exposed USB 1.x, 2.x, and 3.x ports

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

System BIOS or UEFI firmware must by default:

- Support booting through USB 1.x on OHCI controllers
- Support booting through USB 2.x on EHCI controllers
- Support booting through USB 1.x, 2.x, and 3.x on XHCI controllers.
- Support these on all exposed USB ports up to a hub depth of 3.

The system must also support booting Windows PE images from a USB 2.0 device by using extended INT 13 or UEFI native interface in less than 90 seconds.

Design Notes:

OEMs are encouraged to test the boot functionality by creating a bootable USB flash drive with WinPE. See the OPK for details. Vendors may license WinPE (at no charge). For information, send an e-mail to licwinpe@microsoft.com.

System.Fundamentals.USBBoot.SupportSecureStartUpInPreOS

Systems support secure startup by providing system firmware support for writing to and reading from USB flash devices in the pre-operating system environment.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

On all UEFI systems and all systems that implement a TPM, the system must support BitLocker recovery and strong authentication scenarios. This is accomplished by enabling enumeration and full-speed reading/ writing of data (such as key backup and recovery information) from and to a USB mass storage class device, and the UEFI firmware will provide an instance of the block I/O protocol for access to these USB devices.

Design Notes:

See the USB Mass Storage Class Bulk-Only Transport and the USB Mass Storage Class UFI Command specifications, downloadable from http://go.microsoft.com/fwlink/?LinkId=58382.

Send comments about this topic to Microsoft

System.Fundamentals.USBDevice

These requirements apply to USB devices that are integrated into a system.

In this topic:

System.Fundamentals.USBDevice.SelectiveSuspend

All internally connected USB devices must support selective suspend by default.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Selective suspend is an important power saving feature of USB devices. Selective suspension of USB devices is especially useful in portable computers, since it helps conserve battery power.

If a USB device is internally connected, the device driver must enable selective suspend by default. Every USB device driver must place the device into selective suspend within 60 seconds of no user activity. This timeout should be as short as possible while maintaining a good user experience. The selective suspend support can be verified by reviewing the report generated by the powercfg -energy command.

Implementation Notes:

When devices enter selective suspend mode, they are limited to drawing a USB specification defined 2.5mA of current from the USB port. It is important to verify that devices can quickly resume from selective suspend when they are required to be active again.

For example, when selectively suspended, a USB touchpad must detect be able to detect a user's touch and signal resume without requiring the user to press a button. Some devices can lose the ability to detect a wake event when limited to the selective suspend current, 500 microamps per unit load, 2.5mA max. These devices, such as a USB Bluetooth enabled module, must be self-powered, not relying solely on the USB bus for power. By drawing power from another source, the device can detect wake events

For more information about enabling selective suspend for HID devices, please refer to this MSDN article http://msdn.microsoft.com/en-us/library/ff538662(VS.85).aspx

For more information about how to implement selective suspend in a driver, please refer to this white paper:

http://www.microsoft.com/whdc/driver/wdf/USB_select-susp.mspx

To specify a port that is internal (not user visible) and can be connected to an integrated device, the ACPI properties _UPC.PortIsConnectable byte must be set to 0xFF and the _PLD.UserVisible bit must be set to 0. More details are available on MSDN.

Send comments about this topic to Microsoft

System.Fundamentals.WatchDogTimer

A watchdog timer is a device that provides basic watchdog support to a hardware timer exposed by the Microsoft hardware watchdog timer resource table.

In this topic:

System.Fundamentals.WatchDogTimer.IfWatchDogTimerImplemented

If a Watch Dog Timer is implemented and exposed through a WDRT (supported for versions prior to Windows 8) or WDAT (required for Windows 8 and later versions), it must meet Windows compatibility and functionality requirements.

Applies to	Windows 10 x64 Windows 10 x86 Windows Server 2016 Technical Preview x64

Description

Hardware watchdog timer monitors the OS, and reboots the machine if the OS fails to reset the watchdog. The watchdog must meet the requirements and comply with the specification in http://MSDN.microsoft.com/en-us/windows/hardware/gg463320.aspx.

Send comments about this topic to Microsoft

System.Server.Base

Basic requirements for server systems

In this topic:

System.Server.Base.64Bit

A server system can natively run a 64-bit version of Windows Server.

Applies to	Windows Server 2016 Technical Preview x64

Description

A server system must be able to natively support and run a 64-bit Windows Server operating system.

Devices in a server system must also have 64-bit drivers available for 64-bit operation.

System.Server.Base.BMC

Baseboard management controller solution must meet requirements.

Applies to	Windows Server 2016 Technical Preview x64

Description

Baseboard management controller (BMC) hardware that uses the Microsoft provided IPMI driver and service provider must comply with the Intelligent Platform Management Interface (IPMI) Specification, Version 1.5 Document (Revision 1.1, February 20, 2002, 6/1/04 MARKUP) or later.

The BMC must be connected to the system through a Keyboard Controller Style (KCS) Interface. The BMC and its KCS interface must be discoverable through ACPI, as prescribed in Appendix C3 of the IPMI V1.5 specification.

Support for interrupt is optional for the BMC. If interrupt is supported, the BMC:

- Must not be shared with other hardware devices.
- Must support the Set Global Flags command.

The BMC driver must support PnP and Power Management according to the minimum device fundamental requirements defined in the Windows Hardware Compatibility Program requirements.

The driver must be compliant to the kernel-mode driver framework (KMDF) component of the Windows Driver Framework (WDF) for the Microsoft Windows family of operating systems. A legacy driver, for backward compatibility reasons, must be Windows Driver Model (WDM) compliant.

The driver must provide a WMI interface, including Timeout Configuration through RequestResponseEx() which is defined in the ipmidrv.mof file of the WMI interface.

The driver must have support for ACPI Control:

- HARDWARE ID - IPI0001
- COMPATIBLE ID - IPI0001 optional
- _SRV - 1.5 or 2.0 IPMI

- _CRS/_PRS - Format of resources: A single 2-byte or two 1-byte each I/O port or memory mapped I/O

The driver must call the Windows ACPI driver to get the above listed ACPI data.

Support for interrupt is optional in the driver, if supported the IPMI driver must:

- Assign only one interrupt
- Not share interrupts
- Handle disabling of non-communication interrupts that the driver does not fully support through the Set Global Flags command.
- Be capable of handling both communication and non-communication interrupts.

Design Notes:

To prevent interrupt storm, the driver enables BMC interrupt when it starts and disables BMC interrupt supports when stops by using the "Set BMC Global Enables" IPMI command. The field needs to set is the bit [0] - Receive Message Queue interrupt. However, this bit is shared for KCS communication interrupt and KCS non-communication, so the driver needs to be able to properly handle both interrupts.

A KCS communication interrupt is defined as an OBF-generated interrupt that occurs during the process of sending a request message to the BMC and receiving the corresponding response. It's also encountered during the course of processing a GET_STATUS/ABORT control code.

A KCS non-communication interrupt is defined as an OBF-generated interrupt that occurs when the BMC is not in the process of transferring message data or getting error status. This will typically be an interrupt that occurs while the interface is in the IDLE_STATE].

System.Server.Base.BMCDiscovery

Baseboard Management Controller is discoverable and enumerable.

Applies to	Windows Server 2016 Technical Preview x64

Description

A system that has a baseboard management controller (BMC) present must expose it for discovery and enumeration by Windows through Plug-and-Play (PnP) methods appropriate for its device interface. If the BMC is connected to the system through a non-PnP legacy link such as the keyboard controller style (KCS) interface, its resources must be exposed through SMBIOS or ACPI for discovery and enumeration by Windows.

System.Server.Base.Compliance

Server system includes components and drivers that comply with Windows Hardware Certification Program.

Applies to	Windows Server 2016 Technical Preview x64

Description

All buses, devices, and other components in a system must meet their respective Windows Hardware Certification Program requirements and use drivers that are either included with the Windows operating system installation media or that Microsoft has digitally signed.

System.Server.Base.DevicePCIExpress

Server system includes storage and network solutions that use PCI Express architecture

Applies to	Windows Server 2016 Technical Preview x64

Description

A server system must use PCI Express connectivity for all the storage and network devices installed in the system. The devices may either be adapters installed in PCI Express slots or chip down directly connected to the system board. This requirement does not apply to integrated devices that are part of the chipset.

System.Server.Base.ECC

System memory uses ECC or other technology to prevent single-bit errors from causing system failure.

Applies to	Windows Server 2016 Technical Preview x64

Description

Server systems must support error correction code, memory mirroring, or another technology that can detect and correct at least a single-bit memory error. The system memory and cache must be protected with ECC) or other memory protection. All ECC or otherwise protected RAM, visible to the operating system must be cacheable. The solution must be able to detect at least a double-bit error in one word and to correct a single-bit error in one word, where "word" indicates the width in bits of the memory subsystem. A detected error that cannot be corrected must result in a system fault.

System.Server.Base.EnhancedPlatformIntegrityProtectionForCloudServices [If Implemented]

Server supports hardware- and firmware-based enhanced platform integrity protection for cloud services.

Applies to	Windows Server 2016 Technical Preview x64

Description

This is an “If-Implemented”, optional system requirement for a system providing enhanced security for Windows Server. The server platform must support;

UEFI 2.3.1c or later as defined in the following Requirements, and tested by the respective Tests;

- System.Fundamentals.Firmware.TPR.UEFIEncryptedHDD
- System.Fundamentals.Firmware.UEFIBitLocker
- System.Fundamentals.Firmware.UEFIBootEntries
- System.Fundamentals.Firmware.UEFICompatibility
- System.Fundamentals.Firmware.UEFIDefaultBoot
- System.Fundamentals.Firmware.UEFILegacyFallback
- System.Fundamentals.Firmware.Update

SecureBoot as defined in the following Requirements, and tested by the respective Tests;

System.Fundamentals.Firmware.UEFILegacyFallback

System.Fundamentals.Firmware.UEFISecureBoot

The processors in the system are capable of IOMMU, as defined in the following Requirements, or tested by the respective Tests;

System.Server.Virtualization.ProcessorVirtualizationAssist

The system supports TPM 2.0, as defined in the following Requirements, and tested by the respective Tests;

System.Fundamentals.TPM20.EKCerts

System.Fundamentals.TPM20.TPM20

For systems to be awarded the Assurance AQ, the UEFI flag NoPPIClear must be set to TRUE by default. In addition, the NoPPIProvision flag must be set to TRUE by default. There must be a mechanism in UEFI to confirm the settings of these variables and change them. This requirement is in place to allow for total remote management of TPMs out of the box without additional configuration.

For systems to be awarded the Assurance AQ, the UEFI implementation must be compliant with the needs of code integrity. Specifically;

Data pages must be separate from code pages.

Code and Data pages are at page level granularity for alignment.

The same page will not contain both Data [Read or Write] and executable Code.

The memory mappings for what pages are code and data are correct i.e., it is not the case that the whole image is marked as data or code.

This will be accomplished using the correct build options for creating the UEFI binaries. The system must include the GUID the firmware can set to claim compliance with this requirement.

System.Server.Base.HotPlugECN

Server system that supports native Hot Plug functionality meets requirements defined in Hot-Plug ECN No. 31.

Applies to	Windows Server 2016 Technical Preview x64

Description

A server system must meet requirements defined in the PCI Hot-Plug ECN No. 31 if it supports hot-plug of PCI Express devices or adapters; for example as an inherent behavior of a dynamically hardware partitionable design, or in the form of either Express Module or a comparable hot-plug PCI Express I/O option design.

System.Server.Base.NoPATA

Persistent storage devices on servers classified as Hard Disk Drives must not be PATA.

Applies to	Windows Server 2016 Technical Preview x64

Description

Persistent storage devices classified as Hard Disk Drives, either fixed or removable, must not be controlled by any of the following: Parallel Advanced Technology Attachment (also known as Parallel ATA, PATA, IDE, EIDE, or ATAPI) controllers, to include RAID versions of these devices. PATA controllers of any kind may only be connected to CD, DVD or other storage devices not classified as hard disk drives.

Parallel Advanced Technology Attachment (also known as Parallel ATA, PATA, IDE, EIDE, or ATAPI) controllers, to include RAID versions of these devices, do not support the ability to hot remove a hard disk drive from the system should a hard disk drive fail and need to be replaced. This forces the system to be unavailable for long periods.

Parallel Advanced Technology Attachment (also known as Parallel ATA, PATA, IDE, EIDE, or ATAPI) controllers, to include RAID versions of these devices, do not support the ability to hot remove a hard disk drive from the system should a hard disk drive fail and need to be replaced. This forces the system to be unavailable for long periods.

System.Server.Base.OSInstall

Server system includes a method for installing the operating system for emergency recovery or repair.

Applies to	Windows Server 2016 Technical Preview x64

Description

The server system must provide a method for installing the operating system for emergency repair support. The following are examples of possible solutions:

- PXE support
- Internal or externally attached, bootable, rewriteable DVD.

System.Server.Base.PCIAER

Windows Server systems may implement AER (Advanced Error Reporting) as provided by the platform and specified in PCI Express Base Specification version 2.1 and ACPI Specification 3.0b

Applies to	Windows Server 2016 Technical Preview x64

Description

Server System vendors which elect to set the Advanced Error Reporting (AER) bit (0x3) in the PCI _OSC table in the system BIOS must implement the following:

- The _HID value on the root bus of the system must be PNP0A08, so that the operating system can discover the devices are PCI Express and support AER.
- To use PCI AER with Windows the system must report _OSC control in the device and the _SB/PC10 objects in ACPI. The following bits must be enabled:
  - 0x0 - PCI Express Native Hot Plug
  - 0x1 - Hot Plug Control
  - 0x3 - Advanced Error Reporting (AER)
  - 0x4 - PCI Express reliability structure control
- The MCFG ACPI table in PCI Firmware Specification, Revision 3.0b, so that the operating system can access the AER Capability registers in the PCIe extended configuration space.

Design Notes:

This is an If Implemented requirement for Server system vendors. There is no Windows Server 8 requirement to provide AER_OSC to give control to the operating system, and systems may implement a "firmware first" error policy.

System.Server.Base.RemoteManagement

Server system supports remote, headless, out of band management capabilities.

Applies to	Windows Server 2016 Technical Preview x64

Description

Server systems must provide the capability of being managed without the operating system being present, or when the operating system is not fully functional.

The system must provide the following remote, headless, out of band management capabilities:

- Power up the server
- Power off the server
- Reset the server
- Provide access to Windows Server Emergency Management Services on the server
- View system Stop errors on the server

The following are not required if the system is a pedestal/standalone system with 8GB or less max RAM that was logo'd for Windows Server 2003 prior to December 31, 2007:

- Change BIOS settings of the server
- Select which operating system to start on the server

The above capabilities can be provided using any combination of the following methods:

- Serial port console redirection
- Service processor
- BIOS redirection
- Baseboard Management Controller
- Other management device

This requirement addresses the minimum capabilities required for headless server support.

Design Notes:

Console redirection can be forced with a setup command-line switch,

[/emsport:{com1|com2|usebiossettings|off}

/emsbaudrate:baudrate]

EMS Setup, SPCR and EFI or BIOS redirection settings can be configured per the information at http://msdn2.microsoft.com/en-us/library/ms791506.aspx.

See the Microsoft Headless Server and Emergency Services Design specifications and the IPMI specification at http://go.microsoft.com/fwlink/?linkid=36699.

See service processor console redirection details at http://go.microsoft.com/fwlink/?LinkId=58372.

System.Server.Base.ResourceRebalance

Server device drivers must support Resource Rebalance requests

Applies to	Windows Server 2016 Technical Preview x64

Description

A system wide resource rebalance can be executed on Windows Server. One case where this occurs is when a device is dynamically added to a server. Device drivers must honor the resource rebalance flow and the plug and play requests that are dispatched as part of the flow. Device Drivers must queue all IO requests during the resource rebalance operation.

System.Server.Base.ServerRequiredComponents

Server system must include necessary devices and functionality.

Applies to	Windows Server 2016 Technical Preview x64

Description

Server systems must include the following devices or functionality.

Device or Functionality	Requirement	Comments
IO Bus and Devices		
	System.Fundamentals.SystemPCIController.PCIRequirements	Various PCI Specification reqts
	System.Server.Base.HotPlugECN	Hot Plug ECN #31

Memory	2008 – 1 TB, 2008 R2- 2TB, 2012 - 4 TB, Server Next - 4TB (subject to revision at RTM)	Maximum supported memory
	512 MB	Minimum supported memory
	System.Server.Base.ECC	ECC
Processor	System.Server.Virtualization.ProcessorVirtualizationAssist	Hyper-V support
	1.4 GHz 64-bit	Minimum supported processor speed
	2008 – 64, 2008 R2- 256, 2012 - 640, Server Next - 640 Logical Processors (this is subject to revision at RTM)	Maximum supported processor count
Storage		
	System.Server.Base.NoPATA	IDE, EIDE, ATAPI, Parallel ATA not allowed
	System.Fundamentals.Firmware.Boot.SystemWithBootDeviceGreaterThan	No 2.2 TB Boot HD support for BIOS-based systems
Network	Device.Network.LAN.Base	1GigE and minimum off-load requirement
Recovery	System.Server.Base.OSInstall	For example: DVD, PXE

Debug	System.Fundamentals.DebugPort.SystemExposesDebugInterface	For example: COM, USB, 1394
Remote & OOB Mngmt	System.Server.Base.RemoteManagement	For example: BMC, Service Processor adapter
High Precision Timer	System.Fundamentals.HAL.HPETRequired	
SMBIOS	System.Fundamentals.SMBIOS.SMBIOSSpecification	
	System.Server.SMBIOS.SMBIOS	
Video	System.Server.Graphics.WDDM	Use MS-provided driver, or provide either Display only or full WDDM driver
	System.Fundamentals.Graphics.MicrosoftBasicDisplayDriver	Video minimums, 1024 x 768 x 32 bits per pixel, VESA timing and compliance
RemoteFX	System.Fundamentals.Firmware.SystemCanBootWithEitherGraphicsAdapter	BIOS support for multiple adapters
	System.Fundamentals.Graphics.MultipleOperatingMode	Functionality of multiple

		GPUs requirement
Marker file	System.Fundamentals.MarkerFile.SystemIncludesMarkerFile	Marker file for OCA
Single container	System.Client.PCContainer.PCAppearsAsSingleObject	Computer appears as single object in Devices and Printers folder

The following devices or functionality are not required for Server Systems:

- Bus Controllers & Ports:
  - HD Audio
  - Cardbus & PCMCIA
  - IEEE 1394
  - Secure Digital
- Connectivity:
  - Bluetooth
  - Cardbus & PCMCIA
  - ExpressCard
  - IEEE 1394
  - Infrared
  - Parallel [sysfund-0221] & Serial
  - Wireless USB
- Network:
  - ISDN
  - TCP Chimney NIC

- Display:
  - Auxiliary Displays
- Input:
  - Smart Card Reader
- Streaming Media & Broadcast:
  - Broadcast Receiver
  - Decoder
  - Encoder
  - Video Capture
- TPM:
  - TPM. If implemented, must meet requirements.
  - USB write for BitLocker Recovery
- Watchdog Timer (WDT)
- Baseboard Management Controller (BMC)
- Enhanced Power Management Additional Qualification
- Dynamic Partitioning Additional Qualification,
- Fault Tolerance Additional Qualification
- High Availability Additional Qualification
- Power Management concerning S3, S4 and S5 system states support

System.Server.Base.SystemPCIExpress

Server system supports PCI Express natively

Applies to	Windows Server 2016 Technical Preview x64

Description

Server systems are required to support a PCI Express root complex as a connection of the I/O system to the CPU and memory must comply with the requirements defined in the PCI Express 1.0a (or 1.1)

Base Specification and PCI Local Bus Specification, Revision 2.3, or later. If discrepancies exist, the PCI Express Base Specification takes precedence.

Send comments about this topic to Microsoft

System.Server.BMC

In this topic:

System.Server.BMC.OutOfBandRemoteManageability

Server supports out-of-band remote management capabilities.

Applies to	Windows Server 2016 Technical Preview x64

Description

Server hardware supports out-of-band remote management capability, using IPMI 2.0 through a LAN and/or Serial interfaces, as well as in-band manageability through the system KCS interface via the IPMI Driver.

It is not necessary that the server supports the full IPMI 2.0 specification, as only a subset of functionality is required for out-of-band remote manageability.

Enforcement

This is an “If-Implemented” optional system requirement for a server claiming to be remotely manageable out of band using the de facto IPMI standard. This requirement becomes in effect at the release of Windows Server 2016 Technical Preview.

Business Justification

Out-of-band remote manageability through IPMI 2.0 allow different makes and models of server systems running Windows Server 2016 Technical Preview to operate efficiently in a software-defined datacenter and therefore lowering operational costs for the customers where heterogeneous hardware platforms are deployed. In order to achieve this objective, systems must expose this functionality remotely. A server BMC with a dedicated NIC can make this available via IPMI over LAN. A server BMC that only exposes its IPMI functionality through a Serial interface, must be part of a chassis or enclosure that can translate these management operations to a remote operator on the network (for example, through a Chassis Manager).

Send comments about this topic to Microsoft

System.Server.DynamicPartitioning

This feature defines dynamic partitioning requirements of server systems. This feature is not required of all server systems.

In this topic:

System.Server.DynamicPartitioning.Application

Servers that support hardware partitioning must supply partition management software as a Windows application running on a Windows operating system.

Applies to	Windows Server 2016 Technical Preview x64

Description

Servers that support hardware partitioning must provide partition manager software, which provides the user interface administrators will use to configure hardware partitions. This software must be offered as a Windows application running on a Windows operating system.

System.Server.DynamicPartitioning.ApplicationInterface

Servers that support hardware partitioning must supply partition management software that provides a GUI and a scripting capability for partition management.

Applies to	Windows Server 2016 Technical Preview x64

Description

Servers that support hardware partitioning must supply partition management software that includes support for a graphical user interface for manual partition management and a scripting capability for remote or automated partition management.

System.Server.DynamicPartitioning.ConfigurationPersist

Servers that support hardware partitioning must support persistence of hardware partition configuration information across a reboot and power cycle.

Applies to	Windows Server 2016 Technical Preview x64

Description

The hardware partition configuration on a server that supports hardware partitioning must persist across a reboot, hibernate, resume, and power cycle of the partition or the server. This requirement assumes that no partition change was initiated while the partition was down.

System.Server.DynamicPartitioning.Core

Systems that support Dynamic Hardware Partitioning must meet requirements.

Applies to	Windows Server 2016 Technical Preview x64

Description

Systems must meet the requirements listed below and pass the Dynamic Hardware Partitioning test in the Windows Hardware Lab Kit in order to be listed in the Windows Server Catalog as supporting Dynamic Partitioning:

- System.Server.DynamicPartitioning.Application
- System.Server.DynamicPartitioning.ApplicationInterface
- System.Server.DynamicPartitioning.ConfigurationPersist
- System.Server.DynamicPartitioning.ErrorEffect
- System.Server.DynamicPartitioning.Firmware
- System.Server.DynamicPartitioning.HotAddLocal
- System.Server.DynamicPartitioning.HotAddReplace
- System.Server.DynamicPartitioning.HotAddVisual

- System.Server.DynamicPartitioning.HotReplacePU
- System.Server.DynamicPartitioning.PartialHotAdd
- System.Server.DynamicPartitioning.SoftwareStatus
- System.Server.DynamicPartitioning.Subsystem

System.Server.DynamicPartitioning.ErrorEffect

Errors detected in a hardware partition on servers that support hardware partitioning cause no operating system-detectable effects on other partitions.

Applies to	Windows Server 2016 Technical Preview x64

Description

Hardware (which includes firmware) or software errors that occur within the boundary of a hardware partition on a server that supports hardware partitioning must not affect the operating system environment within other hardware partitions.

System.Server.DynamicPartitioning.Firmware

Servers that support hardware partitioning must provide server description and partitioning flows in firmware that comply with the Dynamic Hardware Partitioning Requirements Specification.

Applies to	Windows Server 2016 Technical Preview x64

Description

System firmware on a server that supports hardware partitioning provides the ACPI server description, handshaking during partitioning events, and initialization of hardware that is to be added to a partition and must be provided in compliance with the Hot Replace Flow and Requirements and the Hot Add Flow and Requirements specifications.

For access to these specifications, send e-mail to DPFB@Microsoft.com.

System.Server.DynamicPartitioning.HotAddLocal

Hardware components on a server that supports hardware partitioning that are within a unit that is hot added to a partition cannot be accessible from other hardware partitions.

Applies to	Windows Server 2016 Technical Preview x64

Description

Processors, memory, and I/O components within any unit that is hot added to an existing hardware partition on a server that supports hardware partitioning must not be directly accessible by software running in any other hardware partition.

System.Server.DynamicPartitioning.HotAddReplace

Servers that support hardware partitioning must support hot addition of processors, memory, and I/O and hot replace of processor and memory subsystems.

Applies to	Windows Server 2016 Technical Preview x64

Description

Servers that support hardware partitioning must support hot addition and hot replacement of all operating system-supported component types. Hot-add PU-supported component types are processors, memory, and I/O. Hot replace- supported component types are processors and memory subsystems.

System.Server.DynamicPartitioning.HotAddVisual

Servers that support hardware partitioning must provide visual user indication of the status of hot-add events if no software-based notification is provided.

Applies to	Windows Server 2016 Technical Preview x64

Description

Servers that support one or more hot-add component features must provide a visual indication of the status of each hot-add event if no partition management software is provided.

System.Server.DynamicPartitioning.HotReplacePU

In servers that support dynamic partitioning, hot replacement PUs must have equal and compatible hardware resources to the PU being replaced.

Applies to	Windows Server 2016 Technical Preview x64

Description

A processor or memory PU used as a replacement on a server that supports dynamic partitioning must have equal and compatible hardware resources to the PU being replaced; that is, the same processor type and stepping and the same memory configuration.

System.Server.DynamicPartitioning.PartialHotAdd

Partial success of a hot-add action on a server that supports dynamic partitioning does not affect the stability of the partition or server.

Applies to	Windows Server 2016 Technical Preview x64

Description

Components associated with a hot-add action on a server that supports dynamic partitioning that fails to start (a parked component) must not have a detrimental effect on other components in the PU, partition, or server.

System.Server.DynamicPartitioning.SoftwareStatus

Servers that support hardware partitioning must supply partition management software that provides the user with status for each hot-add or hot-replace event.

Applies to	Windows Server 2016 Technical Preview x64

Description

Servers that support hardware partitioning must supply partition management software. Status of a hot-add or hot-replace event is made available by the Windows operating system in the affected partition. The PM software must provide visual indication of this status to the PM administrator.

System.Server.DynamicPartitioning.Subsystem

On servers that support dynamic partitioning, I/O subsystems are provided in a different partition unit to processors and memory subsystems.

Applies to	Windows Server 2016 Technical Preview x64

Description

To enable success of the hot replace feature, I/O subsystems must be implemented in a different PU to processors and memory subsystems on servers that support dynamic partitioning.

Send comments about this topic to Microsoft

System.Server.FaultTolerant

This feature defines fault tolerant requirements of server systems.

In this topic:

System.Server.FaultTolerant.Core

Systems supporting Fault Tolerant operations must meet requirements.

Applies to	Windows Server 2016 Technical Preview x64

Description

Systems must meet the requirements listed below and pass the Fault Tolerance test in the Windows Hardware Certification Kit in order to be listed in the Windows Server Catalog as having Fault Tolerance.

A Fault Tolerant set [FT set] of systems is a grouping of systems that provide redundancy for every hardware component in a single system of the FT set and can mask any hardware failure such that network-connected clients are not impacted by the hardware failure, such as by loss of connectivity due to network timeout to the FT set due to the host name, domain name, MAC address or IP address, and the services or applications hosted on the FT set, becoming unavailable to those network connected clients. Additionally, an FT set appears to network-connected clients as one system with a single host name, domain name, MAC address or IP address, and unique instances of services or applications.

An FT set must include system clocks that operate in actual lockstep, i.e., there is only one clock domain for the FT set, or virtual lockstep, i.e., the clocks in the systems that comprise the FT set are synchronized at regular intervals of much less than one second. This allows the FT set to always respond to exactly the same interrupts at exactly the same time, and thus be executing exactly the same instructions and have exactly the same state at all times, thus providing the required redundancy.

An FT set is able to resynchronize, i.e., make identical, operating system images after a hardware failure in one system of the FT set is corrected, such that network-connected clients are not impacted by the resynchronization, such as by loss of connectivity due to network timeout to the FT set due to the host name, domain name, MAC address or IP address, and the services or applications hosted on the FT set, becoming unavailable to those network connected clients. The correction of the problem may be by replacement or repair of the failed hardware component, or if the hardware failure is transient, may be cleared by a system reset that forces the re-initialization of all the devices in the system that is part of the FT set.

FT systems may disable or not include devices which could cause asynchronous interrupts to occur such that one system in the FT redundant set had to respond to an interrupt to which the other system(s) of the FT set did not experience. Examples of such devices would be monitoring devices [thermal, voltage, etc.], or external devices that would allow a user to inadvertently interrupt or access only one system in an FT set, such as a CD/DVD device, keyboard, mouse, etc.

Send comments about this topic to Microsoft

System.Server.Firmware.UEFI.GOP

This section describes requirements for systems implementing UEFI firmware.

In this topic:

System.Server.Firmware.UEFI.GOP.Display

System firmware must support GOP and Windows display requirements.

Applies to	Windows Server 2016 Technical Preview x64

Description

If the system firmware supports UEFI, it may choose to additionally support the Graphics Output Protocol (GOP). If the Graphics Output Protocol (GOP) is supported is must be as defined in UEFI 2.3.1.

During this time when the firmware is in control, the following are the requirements:

Topology Selection

- UEFI must reliably detect all the displays that are connected to the POST adapter. The Pre-OS screen can only be displayed on a display connected to the POST adapter.
- In case multiple displays are detected, UEFI must display the Pre-OS screen based on the following logic:
  - System with an Integrated display(Laptop, All In One, Tablet): UEFI must display the Pre-OS screen only on the integrated display
  - System without an Integrated display (integrated display is shut or desktop system): UEFI must display the Pre-OS screen on one display. UEFI must select the display by prioritizing the displays based on connector type. The prioritization is as follows: DisplayPort, HDMI, DVI, HD15, Component, S-Video. If there are multiple monitors connected using the same connector type, the firmware can select which one to use.

Mode Selection

- Once UEFI has determined which display to enabled to display the Pre-OS screen, it must select the mode to apply based on the following logic:
  - System with an Integrated display(Laptop, All In One, Tablet): The display must always be set to its native resolution and native timing
  - System without an Integrated display (desktop):
    - UEFI must attempt to set the native resolution and timing of the display by obtaining it from the EDID.
    - If that is not supported, UEFI must select an alternate mode that matches the same aspect ratio as the native resolution of the display.
    - At the minimum, UEFI must set a mode of 1024 x 768
    - If the display device does not provide an EDID, UEDI must set a mode of 1024 x 768
  - The firmware must always use a 32 bit linear frame buffer to display the Pre-OS screen
  - PixelsPerScanLine must be equal to the HorizontalResolution.
  - PixelFormat must be PixelBlueGreenRedReserved8BitPerColor. Note that a physical frame buffer is required; PixelBltOnly is not supported.

Mode Pruning

- UEFI must prune the list of available modes in accordance with the requirements called out in EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode() (as specified in the UEFI specification version 2.1).

Providing the EDID

- Once the UEFI has set a mode on the appropriate display (based on Topology Selection), UEFI must obtain the EDID of the display and pass it to Windows when Windows uses the EFI_EDID_DISCOVERED_PROTOCOL (as specified in the UEFI specification version 2.1)to query for the EDID.
- It is possible that some integrated panels might not have an EDID in the display panel itself. In this case, UEFI must manufacture the EDID. The EDID must accurately specify the native timing and the physical dimensions of the integrated panel.
- If the display is not integrated and does not have an EDID, then the UEFI does not need to manufacture an EDID.

Send comments about this topic to Microsoft

System.Server.Firmware.VBE

The requirements in this section are enforced on any graphics device with firmware supporting VBE and driver is implementing display portion of the WDDM.

In this topic:

System.Server.Firmware.VBE.Display

System firmware that supports VBE must comply with the Windows Display requirements.

Applies to	Windows Server 2016 Technical Preview x64

Description

If a system firmware supports VBE for display control then it must meet the following requirements:

The display is controlled by the video device firmware before the WDDM graphics driver takes over. During this time when the firmware is in control, the following are the requirements:

Topology Selection

- Video device firmware must reliably detect all the displays that are connected to the POST adapter. The Pre-OS screen can only be displayed on a display connected to the POST adapter.
- In case multiple displays are detected, video device firmware must display the Pre-OS screen based on the following logic:
  - System with an integrated display(Laptop, All In One, Tablet/Convertible): Video device firmware must display the Pre-OS screen only on the integrated display
  - System without an integrated display (integrated display is shut or desktop system): Video device firmware must display the Pre-OS screen on one display. The video device firmware must select the display by prioritizing the displays based on connector type. The prioritization is as follows: DisplayPort, HDMI, DVI, HD15, Component, S-Video. If there are multiple monitors connected using the same connector type, the firmware can select which one to use.

Mode Selection

- Once video device firmware has determined which display to enabled to display the Pre-OS screen, it must select the mode to apply based on the following logic:
  - System with an Integrated display(Laptop, All In One, Tablet/Convertible): The display must always be set to its native resolution and native timing
  - System without an Integrated display (desktop):
    - The video device firmware must attempt to set the native resolution and timing of the display by obtaining it from the EDID
    - If that is not supported, the video device firmware must select an alternate mode that matches the same aspect ratio as the native resolution of the display.
    - At the minimum, the video device firmware must set a mode of 1024 x 768
    - If the display device does not provide an EDID, UEDI must set a mode of 1024 x 768
  - The video device firmware must always use a 32 bit linear frame buffer to display the Pre-OS screen
  - PixelsPerScanLine must be equal to the HorizontalResolution.
  - PixelFormat must be PixelBlueGreenRedReserved8BitPerColor. Note that a physical frame buffer is required; PixelBltOnly is not supported.

Mode Pruning

- The video device firmware must provide a list of modes to Windows when Windows uses the Function 01h (Return VBE Mode Information) as specified in the VESA BIOS Extension Core Functions Standard Version 3.0.
- The video device firmware must prune the modes as appropriate. It should only enumerate the modes that are supported in the EDID of the display that is currently active. It is not required to support all the resolutions supported in the EDID.
- The video device firmware must support 800 x 600 and 1024 x 768
- All modes must be progress at 60 Hz.

Providing the EDID

- Once the video device firmware has set a mode on the appropriate display (based on Topology Selection), video device firmware must obtain the EDID of the display and pass it to Windows when Windows uses command 15h (Display Data Channel) as specified in the VESA BIOS Extension Core Functions Standard Version 3.0:
  - It is possible that some integrated panels might not have an EDID in the display panel itself. In this case, video device firmware must manufacture the EDID. The EDID must accurately specify the native timing and the physical dimensions of the integrated panel
  - If the display is not integrated and does not have an EDID, then the video device firmware does not need to manufacture an EDID

Send comments about this topic to Microsoft

System.Server.Graphics

Base for Graphics on Server Systems

In this topic:

System.Server.Graphics.WDDM

All Windows graphics drivers must be WDDM.

Applies to	Windows Server 2016 Technical Preview x64

Description

The WDDM architecture offers functionality to enable features such as desktop composition, enhanced fault Tolerance, video memory manager, scheduler, cross process sharing of D3D surfaces and so on. WDDM was specifically designed for modern graphics devices that are a minimum of Direct3D 10 Feature Level 9_3 with pixel shader 2.0 or better and have all the necessary hardware features to support the WDDM functionality of memory management, scheduling, and fault tolerance. WDDMv1.3 is required by all systems shipped with Windows 10.

Table below explains the scenario usage for the Graphic driver types.

	Client	Server	Client running in a Virtual Environment	Server Virtual

Full Graphics	Required as post device	Optional	Optional	Optional
Display Only	Not allowed	Optional	Optional	Optional
Render Only	Optional as non primary adapter	Optional	Optional	Optional
Headless	Not allowed	Optional	N/A	N/A

Send comments about this topic to Microsoft

System.Server.PowerManageable

This feature defines power manageable requirements of server systems.

In this topic:

System.Server.PowerManageable.ACPIPowerInterface

Power manageable servers support the power metering and budgeting ACPI interface.

Applies to	Windows Server 2016 Technical Preview x64

Description

Server provides support for reading system level power consumption and reading and writing the system power budget for the server using the 'Power Supply, Metering, and Budgeting Interface' in the ACPI 4.0 specification. The system power budget provides a supported range that the budget can be set to where the minimum budget value is lower than the maximum budget value. The power meter supports a range of averaging intervals such that the minimum averaging interval is one second or lower and the maximum averaging interval is five minutes or higher. To align with the specification, the sampling interval for the power meter must be equal to or less than the minimum averaging interval.

System.Server.PowerManageable.PerformanceStates

If processor(s) in a server system support performance states, the server provides mechanisms to makes these states available to Windows.

Applies to	Windows Server 2016 Technical Preview x64

Description

If the processors on the server support performance states, the server provides firmware mechanisms to pass control of processor performance states to Windows. This mechanism must be enabled by default on the server.

System.Server.PowerManageable.RemotePowerControl

Power manageable server provides a standards based remote out-of-band interface to query and control the power of the system.

Applies to	Windows Server 2016 Technical Preview x64

Description

Power manageable server provides an out of band remote management interface from the server's Baseboard Management Controller (BMC) that is complaint with the IPMI, DCMI, or SMASH (via WS-MAN) 'Power State Management Profile' to query the power state, power on or off (soft off) the server remotely.

This is a requirement for the Power Manageable Additional Qualification for Windows Server.

More detail on the SMASH profile can be found on the Distributed Management task Force web site at - http://www.dmtf.org/standards/published_documents/DSP1027.pdf.

Send comments about this topic to Microsoft

System.Server.RemoteFX

This feature defines RemoteFX requirements of server systems.

In this topic:

System.Server.RemoteFX.RemoteFX

Server systems supporting RemoteFX must meet requirements.

Applies to	Windows Server 2016 Technical Preview x64

Description

Servers must meet the following requirements:

- CPU SLAT- the CPU must support SLAT. This requirement will assist in the performance in the RemoteFX virtualization scenarios.
- GPU requirements - must be WDDM GPUs that support Direct3D11. These requirements apply to only the GPUs intended to support RemoteFX workloads.
- Homogenous GPUs for RemoteFX- workloads - the GPUs that are intended to run RemoteFX workloads must be the same GPU running the same hardware driver.

Send comments about this topic to Microsoft

System.Server.SMBIOS

This feature defines SMBIOS requirements of server systems

In this topic:

System.Server.SMBIOS.SMBIOS

System firmware must fully and accurately implement SMBIOS structures of type 16 and of type 17

Applies to	Windows Server 2016 Technical Preview x64

Description

System firmware must fully and accurately implement SMBIOS structures of type 16 (description of physical memory arrays) and of type 17 (description of memory devices), as permitted by the SMBIOS specification. Implementation of other SMBIOS memory description structures - Types 19, 20 and 37 - are optional.

A JEDEC compliant DRAM DIMM supports Serial Presence Detect (SPD). Through this mechanism and defined standards, the module can be identified in terms of its manufacturer, serial number and other useful information. The JEDEC standards require specific data to reside in the lower 128 bytes of an EEPROM located on the memory module. Programming of this EEPROM is normally done by vendors of DRAM DIMMs at their origin of manufacture, and can optionally be redone afterward to

meet their OEMs' specifications or retailers' requirements for branding purposes while going through distribution channels.

The system firmware (BIOS or UEFI) probes and extracts this information from the DIMM via its SMBus interface. The system firmware uses this information to configure the memory controller. System firmware that supports SMBIOS V2.4 or later, conveys the above DIMM specific information to the operating systems and running applications via a series of SMBIOS structures ("tables") for memory descriptions. These SMBIOS structures also describe the system memory topology, geometry and characteristics. Those are briefly described here for reference purposes and can be found in the current SMBIOS V2.5 Specification (September 5, 2006):

- Physical Memory Array (Type 16) containing information on Location, Use and Error Correction Types; pages 51-52.
- Memory Array Mapped Address (Type 19) containing address mapping for a Physical Memory Array (one structure is present for each contiguous address range described); page 56.
- Memory Device (Type 17) containing information on Form Factor, Type and Type Detail; pages 52-54
- Memory Device Mapped Address (Type 20) containing address mapping to a device-level granularity (one structure is present for each contiguous address range described); page 56.
- Memory Channel (Type 37) containing correlation between a Memory Channel and its associated Memory Devices (each device presents one or more loads to the channel); page 68. This support in the system firmware will:
    - Allow the customers to manage their server memory components as deployed IT assets, and to maintain a comprehensive understanding of their investment of these assets in terms of RAS abilities and cost of ownership.
    - Allow server and data center management solutions to exploit this information in their diagnostic tools and methods for better RAS abilities.
    - Enable certain classes of ISV products (RAM disk, etc.) to exploit this information for better performance and functionalities on Windows platforms.

Send comments about this topic to Microsoft

System.Server.SVVP

This feature defines requirements for the SVVP program.

In this topic:

System.Server.SVVP.SVVP

Products participating in the Server Virtualization Validation Program must meet requirements.

Applies to	Windows Server 2016 Technical Preview x64

Description

Server platforms participating in the Server Virtualization Validation Program must meet the requirements called out here: http://www.windowsservercatalog.com/svvp.aspx.

Send comments about this topic to Microsoft

System.Server.SystemStress

This feature defines system stress requirements of server systems.

In this topic:

System.Server.SystemStress.ServerStress

Server system must function correctly under stress.

Applies to	Windows Server 2016 Technical Preview x64

Description

Server system must operate correctly under long-haul, non-deterministic, high stress conditions. The hardware and software components of the Server system must not cause data corruption, hangs, leaks, memory resource fragmentation, crashes, or have impacts on other components of the system. Server systems must be able to reliably shutdown and restart while under stress to prevent unnecessary and unplanned downtime.

This will be tested using stress tools that emulate loads which may be placed upon a Windows Server system.

Send comments about this topic to Microsoft

System.Server.Virtualization

This feature defines virtualization requirements of server systems.

In this topic:

System.Server.Virtualization.ProcessorVirtualizationAssist

Processors in the server support virtualization hardware assists.

Applies to	Windows Server 2016 Technical Preview x64

Description

All processors in the server must support one of the following processor virtualization hardware assist technologies:

- Intel VT technology
- AMD-V technology

Details on specific requirements for each of these technologies are available in the Windows Server 2008 Virtualization Requirements document.

For access to the Windows Server 2008 Virtualization Requirements document, send e-mail to lhvrtreq@microsoft.com.

Send comments about this topic to Microsoft

System.Server.WHEA

In this topic:

System.Server.WHEA.Core

Server enables reporting of system hardware errors to the operating system.

Applies to	Windows Server 2016 Technical Preview x64

Description

Servers are required to provide mechanisms to enable reporting or communication of corrected and uncorrected system hardware errors that are available on the server to the operating system. The platform may perform thresholding of corrected errors.

The minimal set of error sources are:

- IA64 - Machine Check Exception, Corrected Machine Check, Corrected Platform Error, INIT, PCI Express AER
- X86-64 - Machine Check Exception, Corrected Machine Check, Non Maskable Interrupt, PCI Express AER, BOOT errors.

An interface must be provided on the server to facilitate persistence of error records. The interface must preserve the error records across a server reboot and power cycle. At a minimum, the platform must provide enough storage space for one uncorrectable error record. Options to implement this interface are described in the WHEA Platform Design Guide.

Windows Server provides an OSC mechanism to indicate the presence of WHEA in the OS. The server must honor these mechanisms and enable WHEA flows when the OSC is detected. Optional mechanisms are provided to enable the firmware to process error events from specified error sources before handing control off to WHEA and to communicate this behavior to the OS. This helps avoid conflicts on software handling of hardware error events.These mechanisms are described in the WHEA Platform Design Guide.

Servers must support WHEA-defined interfaces for software insertion of a subset of hardware error conditions into the platform to enable WHEA validation. The injection mechanism must support the injection of one fatal uncorrected and one corrected error; each injectable error is injected using one of the error sources on the platform, and using the signaling mechanism specified for that error source. Options to provide this interface are described in the WHEA Platform Design Guide.

For access to the WHEA Platform Design Guide, send e-mail to WHEAFB@Microsoft.com.

Send comments about this topic to Microsoft

Appendix A: Removed Requirements

The following requirements which were part of the Windows 7, Windows 8, and Windows 8.1 Certification programs will not be part of the Windows 10 Compatibility program. We are reviewing whether or not they are still applicable to Windows 7, Windows 8, or Windows 8.1.

- System.Client.Aero.SystemsStartAeroTheme
- System.Client.BluetoothController.Base.RadioScanIntervalSettings
- System.Client.BluetoothController.Base.SystemsWithBluetoothImplementDeviceID
- System.Client.Buttons.WindowsButtons
- System.Client.CPU.MADT

- System.Client.Digitizer.Base.DigitizersAppearAsHID
- System.Client.Digitizer.Base.HighQualityDigitizerInput
- System.Client.Digitizer.Pen.100HzSampleRate
- System.Client.Digitizer.Pen.ContactAccuracy
- System.Client.Digitizer.Pen.HoverAccuracy
- System.Client.Digitizer.Pen.PenRange
- System.Client.Digitizer.Pen.PenResolution
- System.Client.Digitizer.Touch.5TouchPointMinimum
- System.Client.Digitizer.Touch.DigitizerConnectsOverUSBOrI2C
- System.Client.Digitizer.Touch.DigitizerJitter
- System.Client.Digitizer.Touch.ExtraInputBehavior
- System.Client.Digitizer.Touch.FieldFirmwareUpdatable
- System.Client.Digitizer.Touch.HIDCompliantFirmware
- System.Client.Digitizer.Touch.HighQualityTouchDigitizerInput
- System.Client.Digitizer.Touch.HighResolutionTimeStamp
- System.Client.Digitizer.Touch.InputSeparation
- System.Client.Digitizer.Touch.NoiseSuppression
- System.Client.Digitizer.Touch.PhysicalDimension
- System.Client.Digitizer.Touch.PhysicalInputPosition
- System.Client.Digitizer.Touch.PowerStates
- System.Client.Digitizer.Touch.ReportingRate
- System.Client.Digitizer.Touch.ResponseLatency
- System.Client.Digitizer.Touch.TouchResolution
- System.Client.Digitizer.Touch.ZAxisAllowance
- System.Client.Firewall.FirewallEnabled
- System.Client.Graphics.SingleGPU
- System.Client.Graphics.Windows7.MinimumDirectXLevel
- System.Client.MediaTranscode.GlitchFreeRealtimeCommunication
- System.Client.MediaTranscode.SystemTranscodeFasterThanRealTime
- System.Client.NearFieldProximity.ImplementingProximity
- System.Client.NearFieldProximity.RangeOfActuation

- System.Client.NearFieldProximity.TouchMark
- System.Client.PrecisionTouchpad.PrecisionTouchpad
- System.Client.PrecisionTouchpad.RequiredForARM
- System.Client.RadioManagement.RadioManagerCOMObject
- System.Client.Sensor.GNSSRFSensitivity
- System.Client.Sensor.HumanProximitySensor
- System.Client.Sensor.Integrated
- System.Client.Sensor.Base.ALSCalibrationTest
- System.Client.Sensor.Base.DataEvents
- System.Client.Sensor.Base.GNSSTestProperties
- System.Client.Sensor.Base.PowerState
- System.Client.Sensor.Base.SupportDataTypesAndProperties
- System.Client.Sensor.Base.HID.ReportDescriptor
- System.Client.SpecializedPC.UniqueScenario
- System.Client.SystemConfiguration.SysInfo
- System.Client.SystemConfiguration.Windows7NeccessaryDevices
- System.Client.SystemConfiguration.Windows8RequiredComponents
- System.Client.SystemImage.PushButtonReset
- System.Client.Tablet.ColdBootLatency
- System.Client.Tablet.RequiredHardwareButtons
- System.Client.Tablet.Graphics.MinimumResolution
- System.Client.UMPC.Graphics.WDDM
- System.Client.VideoPlayback.GlitchfreeHDVideoPlayback
- System.Client.VideoPlayback.GlitchfreePlayback
- System.Client.VideoPlayback.WNGlitchfreeHDVideoPlayback
- System.Client.Webcam.Device
- System.Client.Webcam.PhysicalLocation
- System.Client.Webcam.VideoCaptureAndCamera
- System.Client.Webcam.NMSD.NonMSDriver
- System.Client.Webcam.Specification.CameraRequirements
- System.Fundamentals.Firmware.ACPIRequired

- System.Fundamentals.Graphics.MultipleOperatingMode
- System.Fundamentals.Graphics.Windows7.MultipleOperatingModes
- System.Fundamentals.Graphics.Display.MinimumResolutionandColorDepth
- System.Fundamentals.Graphics.DisplayRender.Performance
- System.Fundamentals.ImageVerification.ImageVerification
- System.Fundamentals.PowerManagement.PCResumesInTwoSeconds
- System.Fundamentals.PowerManagement.PCSupportsS3S4S5
- System.Fundamentals.PowerManagement.CS.ConnectedStandby
- System.Fundamentals.PowerManagement.CS.FanOff
- System.Fundamentals.SystemAudio.HardwareVolumeControl
- System.Fundamentals.SystemAudio.NoiseOnTheSignal
- System.Fundamentals.SystemAudio.SystemEmploysAntiPop
- System.Fundamentals.SystemAudio.SystemMicArray
- System.Fundamentals.SystemAudio.3rdPartyDriver.UAA
- System.Fundamentals.SystemPCIController.SystemImplementingRiserCard
- System.Fundamentals.SystemUSB.EHCIToXHCIControllerTransitions
- System.Fundamentals.SystemUSB.SuperSpeedPortsAreVisualDifferent
- System.Fundamentals.SystemUSB.SuperSpeedTerminationRemainsOn
- System.Fundamentals.SystemUSB.USB3andUSB2PortsRoutedToSameXHCIController
- System.Fundamentals.SystemUSB.xHCICompatibleUnlessForApprovedTargetDesigns
- System.Fundamentals.SystemUSB.XHCIControllerSaveState
- System.Fundamentals.SystemUSB.xHCIControllerSupportMSIInterrupts
- System.Fundamentals.SystemUSB.XhciSupportsRuntimePowerManagement
- System.Fundamentals.SystemUSB.XHCIToEHCIControllerTransitions
- System.Fundamentals.TPM.CS.ConnectedStandby
- System.Fundamentals.TPM.NonCS.NonConnectedStandby
- System.Fundamentals.TPM20.TPM20Required
- System.Fundamentals.TrustedPlatformModule.Windows7SystemsTPM
- System.Server.Base.Essentials
- System.Server.Base.PCI23
- System.Server.Graphics.XDDM.No3DSupport

Send comments about this topic to Microsoft