



Windows Hardware Compatibility Specifications for Windows 11, Version 24H2

Disclaimer

This document is provided “as-is”. Information and view expressed in this document, including URL and other Internet website references, may change without notice.

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

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. This document is confidential and proprietary to Microsoft. It is disclosed and can be used only pursuant to a nondisclosure agreement.

Internet Explorer, Microsoft, TechNet, Windows, and Excel are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Table of Contents

Disclaimer.....	2
Glossary.....	39
Device Requirements	39
Device.Audio.APO	39
Device.Audio.APO.BestPractices.....	40
Device.Audio.APO.ClassRegistration	40
Device.Audio.APO.MicArrayRawData.....	40
Device.Audio.APO.RelativeRegistration	40
Device.Audio.APO.ThirdParty	41
Device.Audio.APO.VoiceClarity.Supported	42
Device.Audio.Base.....	43
Device.Audio.Base.AudioProcessing.....	43
Device.Audio.Base.Endpoints	44
Device.Audio.Base.HardwareArchitecture.....	46
Device.Audio.Base.KSTopologyCompliance.....	47
Device.Audio.Base.PowerManagement.....	48
Device.Audio.Base.SamplePositionAccuracy.....	49
Device.Audio.Base.StreamingFormats	49
Device.Audio.Base.VolumeControl.....	50
Device.Audio.Base.WaveRTConformance	51
Device.Audio.HardwareAudioProcessing.....	52
Device.Audio.HardwareAudioProcessing.AudioHardwareOffloading	52
Device.Audio.HardwareAudioProcessing.Loopback	53
Device.Audio.HDAudio	54
Device.Audio.HDAudio.HDAudioSpecCompliance	54
Device.Audio.USB	55
Device.Audio.USB.HIDControls	55
Device.Audio.USB.USB.....	56
Device.BusController	56
Device.BusController.Bluetooth.Base	56
Device.BusController.Bluetooth.Base.4LeSpecification.....	56
Device.BusController.Bluetooth.Base.LeWhiteList	57
Device.BusController.Bluetooth.Base.MicrosoftBluetoothStack	57
Device.BusController.Bluetooth.Base.NoBluetoothLEFilterDriver	57

Device.BusController.Bluetooth.Base.OnOffStateControllableViaSoftware	58
Device.BusController.Bluetooth.Base.Scatternet	58
Device.BusController.Bluetooth.Base.SimultaneousBrEdrAndLeTraffic	58
Device.BusController.Bluetooth.Base.SpecificInformationParameters	59
Device.BusController.Bluetooth.Base.SupportsBluetooth21AndEdr	59
Device.BusController.Bluetooth.WidebandSpeech	59
Device.BusController.Bluetooth.Base.Audio	60
Device.BusController.Bluetooth.Base.Audio.CoreRequirement	60
Device.BusController.Bluetooth.CoreSpecification4Dot1.LELinkLayerTopology	60
Device.BusController.Bluetooth.CoreSpecification4Dot1.LELinkLayerTopology.CoreRequirement	60
Device.BusController.Bluetooth.CoreSpecification5.LEExtendedAdvertising	61
Device.BusController.Bluetooth.CoreSpecification5.LEExtendedAdvertising.CoreRequirement	61
Device.BusController.Bluetooth.CoreSpecification5Dot2.AutoConnectionEstablishmentProcedure	61
Device.BusController.Bluetooth.CoreSpecification5Dot2.AutoConnectionEstablishmentProcedure.CoreRequirement	61
Device.BusController.Bluetooth.CoreSpecification5Dot2.LEPowerControl	62
Device.BusController.Bluetooth.CoreSpecification5Dot2.LEPowerControl.CoreRequirement	62
Device.BusController.Bluetooth.CoreSpecification5Dot3.LEEnhancedConnectionUpdate	62
Device.BusController.Bluetooth.CoreSpecification5Dot3.LEEnhancedConnectionUpdate.CoreRequirement	62
Device.BusController.Bluetooth.CoreSpecification5Dot3.LEChannelClassification	63
Device.BusController.Bluetooth.CoreSpecification5Dot3.LEChannelClassification.CoreRequirement	63
Device.BusController.Bluetooth.ErrorRecovery	63
Device.BusController.Bluetooth.ErrorRecovery.CoreRequirement	63
Device.BusController.Bluetooth.HCIExtensions1	64
Device.BusController.Bluetooth.HCIExtensions1.CoreRequirement	64
Device.BusController.Bluetooth.HciExtensions.A2dpOffload	65
Device.BusController.Bluetooth.HciExtensions.A2dpOffload.Initialization	65
Device.BusController.Bluetooth.HciExtensions.A2dpOffload.Streaming	65
Device.BusController.Bluetooth.HciExtensions.A2dpOffload.ReportedCodecCapabilities	65
Device.BusController.Bluetooth.HciExtensions.A2dpOffload.SimultaneousSco	66
Device.BusController.Bluetooth.HciExtensions.MonitorAdvertisementV2	66
Device.BusController.Bluetooth.HciExtensions.MonitorAdvertisementV2.CoreRequirement	66
Device.BusController.Bluetooth.LEAudioBroadcast	66
Device.BusController.Bluetooth.LEAudioBroadcast.LEAudioUnicast	67
Device.BusController.Bluetooth.LEAudioBroadcast.PeriodicAdvertising	67
Device.BusController.Bluetooth.LEAudioBroadcast.BroadcastIsochronousStreams	67

Device.BusController.Bluetooth.LEAudioUnicast	68
Device.BusController.Bluetooth.LEAudioUnicast.HciExtensionsMonitorAdvertisementV2	68
Device.BusController.Bluetooth.LEAudioUnicast.BRSecureConnections	68
Device.BusController.Bluetooth.LEAudioUnicast.LEConnectedIsochronousStreams	68
Device.BusController.Bluetooth.LEAudioUnicast.LEPowerControl	69
Device.BusController.Bluetooth.LEAudioUnicast.LEEnhancedConnectionUpdate	69
Device.BusController.Bluetooth.LEAudioUnicast.LEChannelClassification	69
Device.BusController.Bluetooth.LEAudioUnicast.CoreSpecification5Dot3	70
Device.BusController.Bluetooth.LEAudioUnicast.Transport	70
Device.BusController.Bluetooth.MicrosoftStack	70
Device.BusController.Bluetooth.MicrosoftStack.BluetoothDeviceClass	70
Device.BusController.Bluetooth.NonUSB	70
Device.BusController.Bluetooth.NonUSB.Performance	70
Device.BusController.Bluetooth.NonUSB.ScoSupport	71
Device.BusController.Bluetooth.USB	71
Device.BusController.Bluetooth.USB.ScoDataTransportLayer	71
Device.BusController.Bluetooth.HciExtensions.EnhancedControllerDiagnostics	71
Device.BusController.Bluetooth.HciExtensions.EnhancedControllerDiagnostics.CoreRequirement	72
Device.BusController.Bluetooth.UsbTransport.Transport	72
Device.BusController.Bluetooth.UsbTransport.Transport.UsbDescriptors	72
Device.BusController.Bluetooth.UsbTransport.AlternateSetting6	73
Device.BusController.Bluetooth.UsbTransport.AlternateSettings6.CoreRequirement	73
Device.BusController.I2C	73
Device.BusController.I2C.CancellationOfIO	73
Device.BusController.I2C.ClockStretching	73
Device.BusController.I2C.HCKTestability	74
Device.BusController.I2C.IdlePowerManagement	74
Device.BusController.I2C.LockUnlockIOCTL	74
Device.BusController.I2C.NACK	75
Device.BusController.I2C.SPBRead	75
Device.BusController.I2C.SPBSequenceIOCTL	75
Device.BusController.I2C.SPBWrite	76
Device.BusController.I2C.Stress	76
Device.BusController.I3C	76
Device.BusController.I3C.DirectMemoryAddressingIO	77
Device.BusController.I3C.HLKTestability	77

Device.BusController.I3C.IdlePowerManagement	77
Device.BusController.I3C.MIPIConformanceTests.....	78
Device.BusController.I3C.RemoteWake.....	78
Device.BusController.I3C.SleepTests	78
Device.BusController.I3C.SystemPowerManagement	78
Device.BusController.I3C.TestedUsingMicrosoftI3CDrivers	79
Device.BusController.I3C.I2C	79
Device.BusController.I3C.I2C.CancellationOfIO	79
Device.BusController.I3C.I2C.LockUnlockIOCTL.....	79
Device.BusController.I3C.I2C.NACK	79
Device.BusController.I3C.I2C.SPBCx.....	80
Device.BusController.I3C.I2C.SPBRead	80
Device.BusController.I3C.I2C.SPBSequenceIOCTL	81
Device.BusController.I3C.I2C.SPBWrite.....	81
Device.BusController.I3C.I2C.Stress	81
Device.BusController.I3C.I2C.ValidI2C.....	82
Device.BusController.I3C.I3C	82
Device.BusController.I3C.I3C.CancellationOfIO	82
Device.BusController.I3C.I3C.DynamicAddressing	82
Device.BusController.I3C.I3C.InBandInterrupts	83
Device.BusController.I3C.I3C.MaxReadWriteLength	83
Device.BusController.I3C.I3C.NACK	83
Device.BusController.I3C.I3C.NoActiveControllerHandoff	83
Device.BusController.I3C.I3C.Read	84
Device.BusController.I3C.I3C.Sequence	84
Device.BusController.I3C.I3C.StaticAddressing	84
Device.BusController.I3C.I3C.Stress	85
Device.BusController.I3C.I3C.TargetReset	85
Device.BusController.I3C.I3C.Write	85
Device.BusController.NFC	86
Device.BusController.NFC.NearFieldProximity	86
Device.BusController.NFC.NearFieldProximity.Base.....	86
Device.BusController.NFC.NearFieldProximity.NDEF.....	86
Device.BusController.NFC.NearFieldProximity.PublishSubscribe	86
Device.BusController.NFC.NearFieldProximity.TagOperation.....	86
Device.BusController.NFC.RadioManagement.....	87

Device.BusController.NFC.RadioManagement.Base	87
Device.BusController.NFC.CardEmulation.....	87
Device.BusController.NFC.CardEmulation.Base	87
Device.BusController.NFC.CardEmulation.Event	87
Device.BusController.NFC.CardEmulation.Communication	88
Device.BusController.NFC.CardEmulation.RoutingTable	88
Device.BusController.NFC.SmartCard	88
Device.BusController.NFC.SmartCard.ResourceManagement	88
Device.BusController.NFC.SmartCard.Communication.....	89
Device.BusController.SdioController	89
Device.BusController.SdioController.ComplyWithIndustrySpec	89
Device.BusController.SdioController.WdfKmdfDriver	89
Device.BusController.UART	89
Device.BusController.UART.Cancellation	89
Device.BusController.UART.DMA.....	90
Device.BusController.UART.FlowControl	90
Device.BusController.UART.FlushFIFO	90
Device.BusController.UART.GuardBuffer	90
Device.BusController.UART.HCKTestability.....	91
Device.BusController.UART.IdlePowerManagement	91
Device.BusController.UART.Performance	91
Device.BusController.UART.ReadWrite.....	92
Device.BusController.UART.Stress	92
Device.BusController.UART.SupportedBaudRates	92
Device.BusController.UsbController	92
Device.BusController.UsbController.ImplementAtLeastOneXhciSpcStructForUSB2	92
Device.BusController.UsbController.MaintainDeviceStateOnResumeS1andS3.....	93
Device.BusController.UsbController.MustResumeWithoutForcedReset	94
Device.BusController.UsbController.PreserveDeviceStateAfterDisableEnable.....	94
Device.BusController.UsbController.UsbifCertification.....	94
Device.BusController.UsbController.TestedUsingMicrosoftUsbStack	95
Device.BusController.UsbController.XhciAc64Bit	95
Device.BusController.UsbController.XhciAddInCardsMapPortsConsistently	95
Device.BusController.UsbController.XhciAddInCardsReportInternalDevices	97
Device.BusController.UsbController.XhciSupportDebuggingOnAllExposedPorts.....	98
Device.BusController.UsbController.XhciSupportMsiMsiInterrupts	98

Device.BusController.UsbController.XhciSupportsMinimum31Streams.....	98
Device.BusController.UsbController.XhciSupportsRuntimePowerManagement	99
Device.BusController.UsbController.XhciVersionCompliant	99
Device.BusController.UsbController.USB4	100
Device.BusController.UsbController.USB4.DomainSleep	100
Device.BusController.UsbController.USB4.HostRouterSupportsECNsFromUsbif	100
Device.BusController.UsbController.USB4.HostRouterSupportsPCleTunneling.....	100
Device.BusController.UsbController.USB4.HostRouterSupportsThunderbolt3Compatibility.....	100
Device.BusController.UsbController.USB4.TestedUsingMicrosoftUsbStack	101
Device.BusController.UsbController.USB4.DiscreteControllersSupportFPB	101
Device.BusController.UsbController.USBC.....	101
Device.BusController.UsbController.USBC.UsbTypeCAItModeCertification	101
Device.BusController.UsbController.USBC.UsbTypeCUCM	102
Device.BusController.UsbController.USBC.UsbTypeCUCMTCPci	103
Device.BusController.UsbController.USBC.UsbTypeCUCSI	103
Device.Connectivity.BluetoothDevices	104
Device.Connectivity.BluetoothDevices.BluetoothDeviceIdProfileVer13	104
Device.Connectivity.BluetoothDevices.BluetoothHidLimitedDiscoverableMode	104
Device.Connectivity.BluetoothDevices.ComplementarySubsystemList	104
Device.Connectivity.BluetoothDevices.HidInitiatedReconnect.....	105
Device.Connectivity.BluetoothDevices.KeyboardsSupportPasskeyAuthentication.....	105
Device.Connectivity.BluetoothDevices.SupportBluetooth21	105
Device.Connectivity.Network.VerticalPairing.....	105
Device.Connectivity.Network.VerticalPairing.WCN	105
Device.Connectivity.UsbDevices	106
Device.Connectivity.UsbDevices.DebugCompliesWithDebugSpec	107
Device.Connectivity.UsbDevices.DebugCompliesWithDebugSpecUSB3	107
Device.Connectivity.UsbDevices.UsbifCertification	107
Device.Connectivity.UsbDevices.DeviceAttachLessThan100ms	107
Device.Connectivity.UsbDevices.FunctionSuspendSelectiveSuspend	108
Device.Connectivity.UsbDevices.InternalDevicesMustSupportSuspend	108
Device.Connectivity.UsbDevices.IsochronousDeviceAndDriver	109
Device.Connectivity.UsbDevices.MsOsContainerId.....	109
Device.Connectivity.UsbDevices.MustBeFunctionalAfterResume	110
Device.Connectivity.UsbDevices.MustNotDisconnectDuringSuspend.....	110
Device.Connectivity.UsbDevices.MustResumeWithoutForcedReset.....	110

Device.Connectivity.UsbDevices.MustSignalAttachWithin500ms	111
Device.Connectivity.UsbDevices.MustSupportSuspend	111
Device.Connectivity.UsbDevices.RespondAllStringRequests	112
Device.Connectivity.UsbDevices.ResponsesLimitedByWLengthField	112
Device.Connectivity.UsbDevices.SerialNumbers	112
Device.Connectivity.UsbDevices.SerialNumbersUseValidCharacters.....	113
Device.Connectivity.UsbDevices.SuperSpeedOnConnectViaUsb3Port.....	113
Device.Connectivity.UsbDevices.TestedUsingMicrosoftUsbStack.....	113
Device.Connectivity.UsbDevices.UseUsbClassOnlyForControllerOrHub.....	114
Device.Connectivity.UsbDevices.WirelessUsbObtainsWusbLogoFromUsbif	114
Device.Connectivity.UsbDevices.WirelessUsbWiMediaAlliace.....	114
Device.Connectivity.UsbDevices.Discretionary	115
Device.Connectivity.UsbDevices.USBC	115
Device.Connectivity.UsbDevices.USBC.USBTypeCAItModeCertification	115
Device.Connectivity.UsbDevices.USBC.USBTypeCCertifiedCables	115
Device.Connectivity.UsbDevices.USBC.USBTypeCAItModeDeviceCompat	116
Device.Connectivity.UsbHub	116
Device.Connectivity.UsbHub.IdentifyNumOfUserAccessiblePorts	116
Device.Connectivity.UsbHub.SupportSuspend	116
Device.Connectivity.UsbHub.Usb3HubCompliesWithUsb3Spec	117
Device.Connectivity.UsbHub.Usb3ReportPortStatusBitsCorrectly	117
Device.Connectivity.UsbHub.USBC	117
Device.Connectivity.UsbHub.USBC.USBTypeCHubCompat.....	117
Device.Connectivity.WSD	118
Device.Connectivity.WSD.DPWS	118
Device.Connectivity.WSD.DPWSExtensibility.....	118
Device.Connectivity.WSD.MetadataExchange.....	119
Device.Connectivity.WSD.MetadataValid	119
Device.Connectivity.WSD.Schema	119
Device.Connectivity.WSD.WSDDiscovery	120
Device.DevFund	120
Device.DevFund.DeviceGuard	120
Device.DevFund.DeviceGuard.DriverCompatibility	120
Device.DevFund.DriverFramework.KMDF	121
Device.DevFund.DriverFramework.KMDF.Reliability	121
Device.DevFund.DriverFramework.KMDF.WDFProperINF.....	122

Device.DevFund.DriverFramework.UMDF	123
Device.DevFund.DriverFramework.UMDF.Reliability	123
Device.DevFund.DriverFramework.UMDF.WDFProperINF	124
Device.DevFund.DriverSecurity	125
Device.DevFund.DriverSecurity.DriverPackage	125
Device.DevFund.Security.DriverAttributes	126
Device.DevFund.Firmware	127
Device.DevFund.Firmware.UpdateDriverPackage	127
Device.DevFund.HALExtension	127
Device.DevFund.HALExtension.HAL	127
Device.DevFund.Memory	128
Device.DevFund.Memory.DriverFootprint	128
Device.DevFund.Memory.NXPool	129
Device.DevFund.Reliability	129
Device.DevFund.Reliability.BasicReliabilityAndPerformance	129
Device.DevFund.Reliability.BasicSecurity	130
Device.DevFund.Reliability.BootDriverEmbeddedSignature	131
Device.DevFund.Reliability.DriverInstallUninstallReinstall	132
Device.DevFund.Reliability.DriverUninstallInstallOtherDeviceStability	133
Device.DevFund.Reliability.NoReplacingSysComponents	133
Device.DevFund.Reliability.NormalOpWithDEP	133
Device.DevFund.Reliability.PnPIDs	134
Device.DevFund.Reliability.PnPIRPs	135
Device.DevFund.Reliability.ProperINF	136
Device.DevFund.Reliability.RemoteDesktopServices	137
Device.DevFund.Reliability.PCSupportsLowPowerStates	137
Device.DevFund.Reliability.Signable	138
Device.DevFund.Reliability.3rdParty	139
Device.DevFund.Reliability.3rdParty.FormerTests	139
Device.DevFund.Reliability.Interrupts	139
Device.DevFund.Reliability.Interrupts.BasicReliabilityAndPerformance	139
Device.DevFund.ReliabilityDisk	140
Device.DevFund.ReliabilityDisk.IOCompletionCancellation	140
Device.DevFund.Rollback	141
Device.DevFund.Rollback.Driver	141
Device.DevFund.Security	141

Device.DevFund.Security.AntiMalware	141
Device.Devfund.Security.CodeObfuscation	141
Device.DevFund.Security.HardwareEnforcedStackProtection	142
Device.DevFund.Security.NoTDIFilterAndLSP	142
Device.Devfund.Security.SoftwareBillofMaterials	143
Device.DevFund.StaticTools	143
Device.DevFund.StaticTools.StaticAnalysisForDrivers	143
Device.DevFund.Driver	144
Device.DevFund.Driver.Componentized	144
Device.DevFund.Driver.NoGraphicalUI	144
Device.DevFund.Driver.Runtime.FilePaths	145
Device.DevFund.Driver.Runtime.RegistryPaths	145
Device.DevFund.INF	146
Device.DevFund.INF.Componentized	146
Device.DevFund.INF.Declarative	146
Device.DevFund.INF.FilePaths	147
Device.DevFund.INF.RegistryPaths	149
Device.DevFund.INF.State	150
Device.DevFund.INF.Syntax	151
Device.Display.Monitor	152
Device.Display.Monitor.Base	152
Device.Display.Monitor.DigitalLinkProtection	152
Device.Display.Monitor.EDID	152
Device.Display.Monitor.Modes	153
Device.Display.Monitor.Stereoscopic3DModes	154
Device.Graphics	154
Device.Graphics.AdapterBase	154
Device.Graphics.AdapterBase.ApplicationVerifier	154
Device.Graphics.AdapterBase.DriverVersion	155
Device.Graphics.AdapterBase.PowerManagementCompliance	157
Device.Graphics.AdapterBase.RunFromDriverStore	158
Device.Graphics.AdapterBase.SubsystemResettable	158
Device.Graphics.AdapterBase.Symbols	159
Device.Graphics.AdapterCore	159
Device.Graphics.AdapterCore.DXCoreAttributes	159
Device.Graphics.AdapterCore.D3D12.Metacommands.MachineLearning	159

Device.Graphics.AdapterRender	160
Device.Graphics.AdapterRender.MinimumDirectXLevel	160
Device.Graphics.AdapterRender.RGBFrameBuffer	160
Device.Graphics.AdapterRender.YUVSupport	160
Device.Graphics.AdapterRender.D3D101Core	161
Device.Graphics.AdapterRender.D3D101Core.D3D101CorePrimary	161
Device.Graphics.AdapterRender.D3D101WDDM11	161
Device.Graphics.AdapterRender.D3D101WDDM11.D3D101v11Primary	161
Device.Graphics.AdapterRender.D3D101WDDM12	162
Device.Graphics.AdapterRender.D3D101WDDM12.D3D101v12Primary	162
Device.Graphics.AdapterRender.D3D10ComputeShader	162
Device.Graphics.AdapterRender.D3D10ComputeShader.D3D10CoreC	162
Device.Graphics.AdapterRender.D3D10Core	163
Device.Graphics.AdapterRender.D3D10Core.D3D10CorePrimary	163
Device.Graphics.AdapterRender.D3D10D3D11LogicOps	163
Device.Graphics.AdapterRender.D3D10D3D11LogicOps.D3D10CoreD	163
Device.Graphics.AdapterRender.D3D10Multisampling4X	164
Device.Graphics.AdapterRender.D3D10Multisampling4X.D3D10CoreA	164
Device.Graphics.AdapterRender.D3D10Multisampling8X	164
Device.Graphics.AdapterRender.D3D10Multisampling8X.D3D10CoreB	164
Device.Graphics.AdapterRender.D3D10WDDM11	165
Device.Graphics.AdapterRender.D3D10WDDM11.D3D10v11Primary	165
Device.Graphics.AdapterRender.D3D10WDDM12	165
Device.Graphics.AdapterRender.D3D10WDDM12.D3D10v12Primary	165
Device.Graphics.AdapterRender.D3D10WDDM12.Stereoscopic3DArraySupport	166
Device.Graphics.AdapterRender.D3D111Core	167
Device.Graphics.AdapterRender.D3D111Core.D3D111CorePrimary	167
Device.Graphics.AdapterRender.D3D11ASTC	167
Device.Graphics.AdapterRender.D3D11ASTC.CoreRequirement	167
Device.Graphics.AdapterRender.D3D11ConservativeRasterization	168
Device.Graphics.AdapterRender.D3D11ConservativeRasterization.CoreRequirement	168
Device.Graphics.AdapterRender.D3D11Core	169
Device.Graphics.AdapterRender.D3D11Core.D3D11CorePrimary	169
Device.Graphics.AdapterRender.D3D11DoublePrecisionShader	169
Device.Graphics.AdapterRender.D3D11DoublePrecisionShader.D3D11CoreC	169
Device.Graphics.AdapterRender.D3D11DriverCommandLists	169

Device.Graphics.AdapterRender.D3D11DriverCommandLists.D3D11CoreB	169
Device.Graphics.AdapterRender.D3D11DriverConcurrentObjectCreation.....	170
Device.Graphics.AdapterRender.D3D11DriverConcurrentObjectCreation.D3D11CoreA	170
Device.Graphics.AdapterRender.D3D11Level9WDDM13	170
Device.Graphics.AdapterRender.D3D11Level9WDDM13.Level9Instancing	170
Device.Graphics.AdapterRender.D3D11PartialPrecision	171
Device.Graphics.AdapterRender.D3D11PartialPrecision.D3D11CoreE	171
Device.Graphics.AdapterRender.D3D11RasterizerOrderedViews	171
Device.Graphics.AdapterRender.D3D11RasterizerOrderedViews.CoreRequirement	171
Device.Graphics.AdapterRender.D3D11StencilReference	171
Device.Graphics.AdapterRender.D3D11StencilReference.CoreRequirement	171
Device.Graphics.AdapterRender.D3D11TypedUAVLoads	172
Device.Graphics.AdapterRender.D3D11TypedUAVLoad.CoreRequirement	172
Device.Graphics.AdapterRender.D3D11WDDM12	172
Device.Graphics.AdapterRender.D3D11WDDM12.D3D11v12Primary	172
Device.Graphics.AdapterRender.D3D11WDDM12DoublePrecisionShader	172
Device.Graphics.AdapterRender.D3D11WDDM12DoublePrecisionShader.D3D11v12C	173
Device.Graphics.AdapterRender.D3D11WDDM21	173
Device.Graphics.AdapterRender.D3D11WDDM21.CoreRequirement	173
Device.Graphics.AdapterRender.D3D12ASTC	174
Device.Graphics.AdapterRender.D3D12ASTC.CoreRequirement	174
Device.Graphics.AdapterRender.D3D12ConservativeRasterization	175
Device.Graphics.AdapterRender.D3D12ConservativeRasterization.CoreRequirement	175
Device.Graphics.AdapterRender.D3D12Core	176
Device.Graphics.AdapterRender.D3D12Core.CoreRequirement	176
Device.Graphics.D3D12.DirectSRMetacommands.....	179
Device.Graphics.D3D12.DirectSRMetacommands.Base	179
Device.Graphics.AdapterRender.D3D12.InvertedViewports	179
Device.Graphics.AdapterRender.D3D12.InvertedViewportDepthFlipsZ.CoreRequirement.....	179
Device.Graphics.AdapterRender.D3D12.InvertedViewportHeightFlipsY.CoreRequirement.....	180
Device.Graphics.AdapterRender.D3D12.GPUUploadHeaps	180
Device.Graphics.AdapterRender.D3D12.GPUUploadHeaps.CoreRequirement	180
Device.Graphics.AdapterRender.D3D12Multiadapter	180
Device.Graphics.AdapterRender.D3D12Multiadapter.CoreRequirement	180
Device.Graphics.AdapterRender.D3D12.NonNormalizedSampling	181
Device.Graphics.AdapterRender.D3D12. NonNormalizedSampling.CoreRequirement.....	181

Device.Graphics.AdapterRender.D3D12RasterizerOrderedViews	182
Device.Graphics.AdapterRender.D3D12RasterizerOrderedViews.CoreRequirement	182
Device.Graphics.AdapterRender.D3D12StencilReference	182
Device.Graphics.AdapterRender.D3D12StencilReference.CoreRequirement	182
Device.Graphics.AdapterRender.D3D12TypedUAVLoads	183
Device.Graphics.AdapterRender.D3D12TypedUAVLoad.CoreRequirement	183
Device.Graphics.AdapterRender.D3D12.FormatListCasting	183
Device.Graphics.AdapterRender.D3D12.FormatListCasting.CoreRequirement	183
Device.Graphics.AdapterRender.D3D12.EnhancedBarriers	184
Device.Graphics.AdapterRender.D3D12.EnhancedBarriers.CoreRequirement	184
Device.Graphics.AdapterRender.D3D12.IndependentStencilRefsAndMasks	185
Device.Graphics.AdapterRender.D3D12.IndependentStencilRefsAndMasks.CoreRequirement	185
Device.Graphics.AdapterRender.D3D12.TriangleFan	185
Device.Graphics.AdapterRender.D3D12.TriangleFan.CoreRequirement	185
Device.Graphics.AdapterRender.D3D12.DynamicPipelineState	185
Device.Graphics.AdapterRender.D3D12.DynamicPipelineState.CoreRequirement	185
Device.Graphics.AdapterRender.D3D12.NonNormalizedSampling	187
Device.Graphics.AdapterRender.D3D12.NonNormalizedSampling.CoreRequirement	187
Device.Graphics.AdapterRender.D3D11TiledResource	188
Device.Graphics.AdapterRender.D3D11TiledResource.D3D111CoreA	188
Device.Graphics.AdapterRender.D3D12.TextureCopyBetweenDimensions.CoreRequirement	188
Device.Graphics.AdapterRender.D3D12.UnrestrictedBufferTextureCopyPitch.CoreRequirement	188
Device.Graphics.AdapterRender.D3D12.UnrestrictedVertexElementAlignment.CoreRequirement	189
Device.Graphics.AdapterRender.D3D12VideoDecoding	189
Device.Graphics.AdapterRender.D3D12VideoDecoding.CoreRequirement	189
Device.Graphics.AdapterRender.D3D12VideoProcessing	190
Device.Graphics.AdapterRender.D3D12VideoProcessing.CoreRequirement	190
Device.Graphics.AdapterRender.D3D12VideoEncoding	190
Device.Graphics.AdapterRender.D3D12VideoEncoding.CoreRequirement	190
Device.Graphics.WDDM26.AdapterRender.D3D12.DXILCore.ShaderModel64	190
Device.Graphics.WDDM27.AdapterRender.D3D12.DXILCore.ShaderModel65	191
Device.Graphics.IndirectDisplay.Wired	191
Device.Graphics.IndirectDisplay.Wired.Base	191
Device.Graphics.MCDM27	192
Device.Graphics.MCDM27.CoreRequirement	192
Device.Graphics.MCDM27.OneCore	192

Device.Graphics.MCDM27.OneCore.CoreRequirement.....	192
Device.Graphics.MCDM27.Containers	193
Device.Graphics.MCDM27.Containers.CoreRequirement.....	193
Device.Graphics.WDDM	193
Device.Graphics.WDDM.Base	193
Device.Graphics.WDDM.Checklist	194
Device.Graphics.WDDM.GPUFenceCommands	195
Device.Graphics.WDDM.Display.....	196
Device.Graphics.WDDM.Display.Base	196
Device.Graphics.WDDM.Display.GammaCorrection	196
Device.Graphics.WDDM.Display.HardwareCaptureValidation.....	196
Device.Graphics.WDDM.Display.HotPlugDetection	197
Device.Graphics.WDDM.Display.I2CSupport.....	197
Device.Graphics.WDDM.Display.Multimon.....	198
Device.Graphics.WDDM.DisplayRender	199
Device.Graphics.WDDM.DisplayRender.Base.....	199
Device.Graphics.WDDM.DisplayRender.DriverSetupCompatible.....	199
Device.Graphics.WDDM.DisplayRender.OutputProtection	199
Device.Graphics.WDDM.DisplayRender.Stability.....	200
Device.Graphics.WDDM.Render.....	201
Device.Graphics.WDDM.Render.Base	201
Device.Graphics.WDDM.Render.VideoProcessing	201
Device.Graphics.WDDM11	203
Device.Graphics.WDDM11.Base	203
Device.Graphics.WDDM11.Display	203
Device.Graphics.WDDM11.Display.Base	203
Device.Graphics.WDDM11.DisplayRender.....	204
Device.Graphics.WDDM11.DisplayRender.Base.....	204
Device.Graphics.WDDM11.DisplayRender.D3D9Overlay	204
Device.Graphics.WDDM11.DisplayRender.D3D9Overlay.D3D9Overlay	204
Device.Graphics.WDDM11.Render.....	204
Device.Graphics.WDDM11.Render.Base	204
Device.Graphics.WDDM11.Render.DXVAHD	205
Device.Graphics.WDDM11.Render.DXVAHD.DXVAHD.....	205
Device.Graphics.WDDM12	205
Device.Graphics.WDDM12.Base	205

Device.Graphics.WDDM12.Display	208
Device.Graphics.WDDM12.Display.Base	208
Device.Graphics.WDDM12.Display.ContainerIDSupport	209
Device.Graphics.WDDM12.Display.DisplayOutputControl	210
Device.Graphics.WDDM12.Display.ModeEnumeration	211
Device.Graphics.WDDM12.Display.PnpStopStartSupport	212
Device.Graphics.WDDM12.Display.ProvideLinearFrameBuffer	214
Device.Graphics.WDDM12.DisplayOnly	214
Device.Graphics.WDDM12.DisplayOnly.Base	214
Device.Graphics.WDDM12.DisplayRender	216
Device.Graphics.WDDM12.DisplayRender.Base	216
Device.Graphics.WDDM12.DisplayRender.ProcessingStereoscopicVideoContent	218
Device.Graphics.WDDM12.DisplayRender.ProcessingStereoscopicVideoContent.ProcessingStereoscopicVideoContent	218
Device.Graphics.WDDM12.DisplayRender.RuntimePowerMgmt	219
Device.Graphics.WDDM12.DisplayRender.RuntimePowerMgmt.RuntimePowerMgmt	219
Device.Graphics.WDDM12.Render	219
Device.Graphics.WDDM12.Render.Base	219
Device.Graphics.WDDM12.Render.D3D11VideoDecoding	221
Device.Graphics.WDDM12.Render.D3D11VideoProcessing	222
Device.Graphics.WDDM12.Render.DirectFlip	223
Device.Graphics.WDDM12.Render.FlipOnVSyncMmIo	224
Device.Graphics.WDDM12.Render.OfferReclaim	224
Device.Graphics.WDDM12.Render.PreemptionGranularity	225
Device.Graphics.WDDM12.Render.PremiumContentPlayback	226
Device.Graphics.WDDM12.Render.TDRResiliency	226
Device.Graphics.WDDM12.Render.UMDLogging	227
Device.Graphics.WDDM12.Render.XPSRasterizationConformance	227
Device.Graphics.WDDM12.RenderOnly	228
Device.Graphics.WDDM12.RenderOnly.Base	228
Device.Graphics.WDDM12.StandbyHibernateFlags	229
Device.Graphics.WDDM12.StandbyHibernateFlags.StandbyHibernateFlags	229
Device.Graphics.WDDM13	230
Device.Graphics.WDDM13.Base	230
Device.Graphics.WDDM13.DisplayRender	231
Device.Graphics.WDDM13.DisplayRender.48HzVideoPlayback	232

Device.Graphics.WDDM13.DisplayRender.IndependentFlip	232
Device.Graphics.WDDM13.DisplayRender.MultiplaneOverlaySupport.....	232
Device.Graphics.WDDM13.DisplayRender.CoolingInterface.....	234
Device.Graphics.WDDM13.DisplayRender.CoolingInterface.ThermalHints	234
Device.Graphics.WDDM13.DisplayRender.WirelessDisplay.....	235
Device.Graphics.WDDM13.DisplayRender.WirelessDisplay.BasicWirelessDisplay	235
Device.Graphics.WDDM13.EnhancedPowerManagement.....	236
Device.Graphics.WDDM13.EnhancedPowerManagement.FState	236
Device.Graphics.WDDM13.EnhancedPowerManagement.VSYNC.....	236
Device.Graphics.WDDM13.Render.....	236
Device.Graphics.WDDM13.Render.CheckDDIBoundaries.....	236
Device.Graphics.WDDM13.Render.GPUNode	237
Device.Graphics.WDDM13.Render.HighPerformanceTimingData	237
Device.Graphics.WDDM13.Render.PresentOverheadOptimization.....	238
Device.Graphics.WDDM13.Render.TrimSupport	239
Device.Graphics.WDDM20	239
Device.Graphics.WDDM20.CoreRequirement	239
Device.Graphics.WDDM20.Core.....	241
Device.Graphics.WDDM20.Core.Base	241
Device.Graphics.WDDM20.DisplayRender.....	242
Device.Graphics.WDDM20.DisplayRender.HardwareContentProtection	242
Device.Graphics.WDDM20.Display.VirtualModeSupport	243
Device.Graphics.WDDM20.Display.VirtualModeSupport.CoreRequirement	243
Device.Graphics.WDDM21	243
Device.Graphics.WDDM21.CoreRequirement	243
Device.Graphics.WDDM21.FlipQueueLatency.....	244
Device.Graphics.WDDM22	244
Device.Graphics.WDDM22.DisplayRender.....	244
Device.Graphics.WDDM22.DisplayRender.4HWID	244
Device.Graphics.WDDM22.DisplayRender.PeriodicFrameNotification	245
Device.Graphics.WDDM22.DisplayRender.DisplayWireFormatInfo.....	245
Device.Graphics.WDDM22.DisplayRender.MultiPlaneOverlay3	246
Device.Graphics.WDDM22.DisplayRender.SetTimings	246
Device.Graphics.WDDM22.DisplayRender.DisplayDetection	247
Device.Graphics.WDDM22.DisplayRender.ReducedBlackScreen	247
Device.Graphics.WDDM22.DisplayRender.HighDynamicRange	248

Device.Graphics.WDDM22.DisplayRender.HighDynamicRange.CoreRequirement.....	248
Device.Graphics.WDDM22.DisplayRender.ColorManagement.....	250
Device.Graphics.WDDM22.DisplayRender.ColorManagement.CoreRequirement.....	250
Device.Graphics.WDDM22.Render.BrowserSecurity.....	250
Device.Graphics.WDDM22.Render.BrowserSecurity.ArbitraryCodeGuard	250
Device.Graphics.WDDM22.Render.BrowserSecurity.ControlFlowGuard	251
Device.Graphics.WDDM22.AdapterRender.D3D11.ParametrizedSwizzlesR1	251
Device.Graphics.WDDM22.AdapterRender.D3D11.ParametrizedSwizzlesR1.CoreRequirements	251
Device.Graphics.WDDM22.AdapterRender.D3D12Core	252
Device.Graphics.WDDM22.AdapterRender.D3D12Core.HeapPrioritiesR1	252
Device.Graphics.WDDM22.AdapterRender.D3D12Core.ParametrizedSwizzlesR1	252
Device.Graphics.WDDM22.AdapterRender.D3D12Core.LateLatch.....	252
Device.Graphics.WDDM22.AdapterRender.D3D12Core.RealtimeCommandQueues	252
Device.Graphics.WDDM22.AdapterRender.D3D12Core.FastPSOComplies	253
Device.Graphics.WDDM22.AdapterRender.D3D12Core.ExsistingSysMemHeaps	253
Device.Graphics.WDDM22.AdapterRender.D3D12.WindowsHolographic	254
Device.Graphics.WDDM22.AdapterRender.D3D12.WindowsHolographic.CoreRequirement	254
Device.Graphics.WDDM22.AdapterRender.D3D12.DepthBoundTest	254
Device.Graphics.WDDM22.AdapterRender.D3D12.DepthBoundTest.CoreRequirement	254
Device.Graphics.WDDM22.AdapterRender.D3D12.ASTC.....	255
Device.Graphics.WDDM22.AdapterRender.D3D12.ASTC.CoreRequirement	255
Device.Graphics.WDDM22.AdapterRender.D3D12.StereoPrimitives	255
Device.Graphics.WDDM22.AdapterRender.D3D12.StereoPrimitives.CoreRequirement	255
Device.Graphics.WDDM22.AdapterRender.D3D12.MVPMRTArrayIndexRouting	255
Device.Graphics.WDDM22.AdapterRender.D3D12.MVPMRTArrayIndexRouting.CoreRequirement	255
Device.Graphics.WDDM22.AdapterRender.D3D12.ProgrammableMSAA.....	256
Device.Graphics.WDDM22.AdapterRender.D3D12.ProgrammableMSAA.CoreRequirement	256
Device.Graphics.WDDM22.AdapterRender.D3D12.9on12.....	256
Device.Graphics.WDDM22.AdapterRender.D3D12.9on12.FloatingPointBehavior.....	256
Device.Graphics.WDDM22.AdapterRender.D3D11D3D12.SystemShaderCache	257
Device.Graphics.WDDM22.AdapterRender.D3D11D3D12.SystemShaderCache.CoreRequirement.....	257
Device.Graphics.WDDM22.AdapterRender.D3D12.DXILCore.ShaderModel60	257
Device.Graphics.WDDM22.AdapterRender.D3D12.DXILCore.ShaderModel60.CoreRequirement.....	257
Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.WaveOps.....	258
Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.WaveOps.CoreRequirement.....	258
Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.Int64.....	258

Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.Int64.CoreRequirement.....	258
Device.Graphics.WDDM23	259
Device.Graphics.WDDM23.AdapterRender.D3D12.DXILCore.ShaderModel61	259
Device.Graphics.WDDM23.AdapterRender.D3D12.DXILCore.ShaderModel61.CoreRequirement.....	259
Device.Graphics.WDDM23.AdapterRender.D3D12.ViewInstancing.....	259
Device.Graphics.WDDM23.AdapterRender.D3D12.ViewInstancing.CoreRequirement.....	259
Device.Graphics.WDDM23.AdapterRender.D3D12.Video	260
Device.Graphics.WDDM23.AdapterRender.D3D12.Video.CoreRequirement.....	260
Device.Graphics.WDDM23.AdapterRender.D3D12.DepthBoundsTest	260
Device.Graphics.WDDM23.AdapterRender.D3D12.DepthBoundsTest.CoreRequirement	260
Device.Graphics.WDDM23.AdapterRender.D3D12.ProgrammableMSAA.....	260
Device.Graphics.WDDM23.AdapterRender.D3D12.ProgrammableMSAA.CoreRequirement	260
Device.Graphics.WDDM24	260
Device.Graphics.WDDM24.CoreRequirements	261
Device.Graphics.WDDM24.Render.....	261
Device.Graphics.WDDM24.Render.D3D11VideoDecoding.....	261
Device.Graphics.WDDM24.Render.....	261
Device.Graphics.WDDM24.Render.CrossAdapterResource	261
Device.Graphics.WDDM24.Render.D3D11.VideoDecoding.....	261
Device.Graphics.WDDM24.Render.D3D12ResourceSerialization	262
Device.Graphics.WDDM24.Render.D3D12VideoDecoding.....	262
Device.Graphics.AdapterRender.D3D11WDDM24.....	262
Device.Graphics.AdapterRender.D3D11WDDM24.eFSE.....	262
Device.Graphics.WDDM24.AdapterRender.D3D11.SharedTexture	263
Device.Graphics.WDDM24.AdapterRender.D3D11.SharedTexture.CoreRequirement	263
Device.Graphics.WDDM24.AdapterRender.D3D12.ContentProtection	263
Device.Graphics.WDDM24.AdapterRender.D3D12.ContentProtection.CoreRequirement	263
Device.Graphics.WDDM24.AdapterRender.D3D12.DXILCore.ShaderModel62	263
Device.Graphics.WDDM24.AdapterRender.D3D12.DXILCore.ShaderModel62.CoreRequirement.....	263
Device.Graphics.WDDM24.AdapterRender.D3D12.Native16Bits	264
Device.Graphics.WDDM24.AdapterRender.D3D12.Native16Bits.CoreRequirement	264
Device.Graphics.WDDM24.AdapterRender.D3D12.PeerToPeerLDAAtomics	264
Device.Graphics.WDDM24.AdapterRender.D3D12.PeerToPeerLDAAtomics.CoreRequirement	264
Device.Graphics.WDDM24.AdapterRender.D3D12.SharedTexture	265
Device.Graphics.WDDM24.AdapterRender.D3D12.SharedTexture.CoreRequirement	265
Device.Graphics.WDDM24.Paravirtualization.....	265

Device.Graphics.WDDM24.Paravirtualization.CoreRequirement	265
Device.Graphics.WDDM25	266
Device.Graphics.WDDM25.AdapterRender.D3D12.Raytracing	266
Device.Graphics.WDDM25.AdapterRender.D3D12.DXILCore.ShaderModel63	266
Device.Graphics.WDDM25.AdapterRender.D3D12.Raytracing.CoreRequirement	266
Device.Graphics.WDDM25.Display.Brightness3	267
Device.Graphics.WDDM25.Display.Brightness3.InstantaneousChange	267
Device.Graphics.WDDM25.Display.Brightness3.LinearRamping	267
Device.Graphics.WDDM25.Display.Brightness3.PreferredMaximumBrightness	267
Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayDithering	267
Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayDithering.CoreRequirement	267
Device.Graphics.WDDM25.Display.HighDynamicRange.SDRWhiteLevel	268
Device.Graphics.WDDM25.Display.HighDynamicRange.SDRWhiteLevel.CoreRequirement	268
Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayColorspaceTransform	268
Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayColorspaceTransform.CoreRequirement	268
Device.Graphics.WDDM25.Display.HighDynamicRange.HDRPixelFormatSupport	269
Device.Graphics.WDDM25.Display.HighDynamicRange.HDRPixelFormatSupport.CoreRequirement	269
Device.Graphics.WDDM25.Display.HighDynamicRange.Brightness3	270
Device.Graphics.WDDM25.Display.HighDynamicRange.Brightness3.CoreRequirement	270
Device.Graphics.WDDM25.Display.HighDynamicRange.DirectFlipSupport	270
Device.Graphics.WDDM25.Display.HighDynamicRange.DirectFlipSupport.CoreRequirement	270
Device.Graphics.WDDM25.Display.HighDynamicRange.NeutralColorProcessingControl	271
Device.Graphics.WDDM25.Display.HighDynamicRange.NeutralColorProcessingControl.CoreRequirement ..	271
Device.Graphics.WDDM25.SRVOnlyTiledResourceTier3	271
Device.Graphics.WDDM25.SRVOnlyTiledResourceTier3.CoreRequirement	271
Device.Graphics.WDDM25.OneCoreUAP	271
Device.Graphics.WDDM25.OneCoreUAP.CoreRequirement	271
Device.Graphics.WDDM26	272
Device.Graphics.WDDM26.OneCore	272
Device.Graphics.WDDM26.OneCore.CoreRequirement	272
Device.Graphics.WDDM26.Containers	272
Device.Graphics.WDDM26.Containers.CoreRequirement	272
Device.Graphics.WDDM26.D3D12.Metacommands	272
Device.Graphics.WDDM26.D3D12.Metacommands.GpuDecompression	273
Device.Graphics.WDDM26.D3D12.Metacommands.MachineLearning	273
Device.Graphics.WDDM26.D3D12.VariableRateShading	273

Device.Graphics.WDDM26.D3D12.VariableRateShading.CoreRequirement	273
Device.Graphics.WDDM26.HardwareMotionVectorExtraction	273
Device.Graphics.WDDM26.HardwareMotionVectorExtraction.CoreRequirement	274
Device.Graphics.WDDM27	274
Device.Graphics.WDDM27.D3D12.SamplerFeedback.....	274
Device.Graphics.WDDM27.D3D12.SamplerFeedback.CoreRequirement.....	274
Device.Graphics.WDDM27.D3D12.MeshShaders.....	274
Device.Graphics.WDDM27.D3D12.MeshShaders.CoreRequirement.....	274
Device.Graphics.WDDM27.D3D12.TrackedWorkloads	275
Device.Graphics.WDDM27.D3D12.TrackedWorkloads.CoreRequirement.....	275
Device.Graphics.WDDM27.AdapterRender.D3D12.ContentProtection	275
Device.Graphics.WDDM27.AdapterRender.D3D12.ContentProtection.CoreRequirement	275
Device.Graphics.WDDM27.ParavirtualizationNonDirectX	275
Device.Graphics.WDDM27.ParavirtualizationNonDirectX.CoreRequirement.....	275
Device.Graphics.WDDM27.Display.Diagnostics.....	276
Device.Graphics.WDDM27.Display.Diagnostics.CoreRequirement.....	276
Device.Graphics.WDDM27.Display.MipiDsi.....	276
Device.Graphics.WDDM27.Display.MipiDsi.CoreRequirement.....	276
Device.Graphics.WDDM28	277
Device.Graphics.WDDM29	277
Device.Graphics.WDDM29.Render.CoreRequirement.....	277
Device.Graphics.WDDM30	277
Device.Graphics.WDDM30.AdapterRender.D3D12.DXILCore.ShaderModel66	277
Device.Graphics.WDDM30.Render.CoreRequirement.....	278
Device.Graphics.WDDM30.DisplayRender.CrossAdapterScanOut.....	278
Device.Graphics.WDDM30.Render.SelfRefreshMemorySupport	278
Device.Graphics.WDDM30.Render.SelfRefreshMemorySupport.CoreRequirement.....	278
Device.Graphics.WDDM30.Paravirtualization.BackwardsCompatvGpuSupport	279
Device.Graphics.WDDM30.Paravirtualization.BackwardsCompatvGpuSupport.CoreRequirement	279
Device.Graphics.WDDM30.WSL	279
Device.Graphics.WDDM30.WSL.CoreRequirement	279
Device.Graphics.WDDM30.DisplayRender.FlexibleDisplayableSurfacePresentation.....	279
Device.Graphics.WDDM30.Display.DisplayPortAux	280
Device.Graphics.WDDM30.Display.DisplayPortAux.CoreRequirement	280
Device.Graphics.WDDM30.Display.DisplayPortSbm.....	280
Device.Graphics.WDDM30.Display.DisplayPortSbm.CoreRequirement.....	280

Device.Graphics.WDDM32.Display.DisplayMux	281
Device.Graphics.WDDM32.Display.DisplayMux.CoreRequirement	281
Device.Graphics.WDDM32.Display.DisplayMux.DynamicSwitching	281
Device.Imaging	281
Device.Imaging.Printer.Base	281
Device.Imaging.Printer.Base.applicationVerifier	281
Device.Imaging.Printer.Base.configurationFiles	282
Device.Imaging.Printer.Base.connectionRecovery	283
Device.Imaging.Printer.Base.deviceCapabilities	283
Device.Imaging.Printer.Base.driverCategory	284
Device.Imaging.Printer.Base.DriverPDC	284
Device.Imaging.Printer.Base.GDLFile	284
Device.Imaging.Printer.Base.infFile	285
Device.Imaging.Printer.Base.printProcessor	286
Device.Imaging.Printer.Base.printRegions	286
Device.Imaging.Printer.Base.printTicket	287
Device.Imaging.Printer.Base.rendering	287
Device.Imaging.Printer.Base.TCPMon	288
Device.Imaging.Printer.Mobile	288
Device.Imaging.Printer.Mobile.PDL	288
Device.Imaging.Printer.USB	289
Device.Imaging.Printer.USB.CompatID	289
Device.Imaging.Printer.USB.JSBidiExtender	290
Device.Imaging.Printer.WSD	290
Device.Imaging.Printer.WSD.CompatID	290
Device.Imaging.Printer.XPS	291
Device.Imaging.Printer.XPS.XPS	291
Device.Imaging.Scanner.Base	292
Device.Imaging.Scanner.Base.dataTransfer	292
Device.Imaging.Scanner.Base.wia20	293
Device.Imaging.Scanner.Base.WIAProperties	294
Device.Imaging.Scanner.WSD	294
Device.Imaging.Scanner.WSD.WSScan	294
Device.Input	295
Device.Input.Digitizer.Base	295
Device.Input.Digitizer.Base.NoiseSuppression	295

Device.Input.Digitizer.Base.HIDCompliant	295
Device.Input.Digitizer.Base.ThirdPartyDrivers	296
Device.Input.Digitizer.Pen	296
Device.Input.Digitizer.Pen.Accuracy	296
Device.Input.Digitizer.Pen.BarrelButton	297
Device.Input.Digitizer.Pen.Buffering	297
Device.Input.Digitizer.Pen.CustomGestures	298
Device.Input.Digitizer.Pen.Eraser	298
Device.Input.Digitizer.Pen.HIDCompliant	299
Device.Input.Digitizer.Pen.HighResolutionTimeStamp	299
Device.Input.Digitizer.Pen.HoverRange	299
Device.Input.Digitizer.Pen.Jitter	300
Device.Input.Digitizer.Pen.NoiseSuppression	300
Device.Input.Digitizer.Pen.Pressure	301
Device.Input.Digitizer.Pen.ReportRate	302
Device.Input.Digitizer.Pen.Resolution	303
Device.Input.Digitizer.Pen.ResponseLatency	303
Device.Input.Digitizer.Pen.ThirdPartyDrivers	304
Device.Input.Digitizer.PrecisionTouchpad	304
Device.Input.Digitizer.PrecisionTouchpad.Accuracy	304
Device.Input.Digitizer.PrecisionTouchpad.Buffering	304
Device.Input.Digitizer.PrecisionTouchpad.Buttons	305
Device.Input.Digitizer.PrecisionTouchpad.ContactTipSwitchHeight	305
Device.Input.Digitizer.PrecisionTouchpad.DeviceTypeReporting	305
Device.Input.Digitizer.PrecisionTouchpad.Dimensions	306
Device.Input.Digitizer.PrecisionTouchpad.FingerSeparation	306
Device.Input.Digitizer.PrecisionTouchpad.HIDCompliant	307
Device.Input.Digitizer.PrecisionTouchpad.HighResolutionTimeStamp	307
Device.Input.Digitizer.PrecisionTouchpad.InputResolution	307
Device.Input.Digitizer.PrecisionTouchpad.Jitter	308
Device.Input.Digitizer.PrecisionTouchpad.MinMaxContacts	308
Device.Input.Digitizer.PrecisionTouchpad.NoiseSuppression	309
Device.Input.Digitizer.PrecisionTouchpad.ReportRate	309
Device.Input.Digitizer.PrecisionTouchpad.ResponseLatency	309
Device.Input.Digitizer.PrecisionTouchpad.SelectiveReporting	310
Device.Input.Digitizer.PrecisionTouchpad.ThirdPartyDrivers	311

Device.Input.Digitizer.Touch.....	311
Device.Input.Digitizer.Touch.Accuracy.....	311
Device.Input.Digitizer.Touch.Buffering.....	311
Device.Input.Digitizer.Touch.CustomGestures.....	312
Device.Input.Digitizer.Touch.FingerSeparation.....	312
Device.Input.Digitizer.Touch.HIDCompliant.....	312
Device.Input.Digitizer.Touch.HighResolutionTimeStamp.....	313
Device.Input.Digitizer.Touch.Jitter.....	313
Device.Input.Digitizer.Touch.MinContactCount.....	314
Device.Input.Digitizer.Touch.NoiseSuppression.....	314
Device.Input.Digitizer.Touch.ReportRate.....	314
Device.Input.Digitizer.Touch.Resolution.....	315
Device.Input.Digitizer.Touch.ResponseLatency.....	315
Device.Input.Digitizer.Touch.ThirdPartyDrivers.....	316
Device.Input.FingerPrintReader.....	316
Device.Input.FingerPrintReader.Base.....	316
Device.Input.FingerPrintReader.Extensions.....	318
Device.Input.FingerPrintReader.ManagementApps.....	318
Device.Input.FingerPrintReader.SensorEngineDB.....	318
Device.Input.FingerPrintReader.WBDI.....	319
Device.Input.HID.....	321
Device.Input.HID.I2CProtocolSpecCompliant.....	321
Device.Input.HID.UsbSpecificationCompliant.....	321
Device.Input.Keyboard.....	322
Device.Input.Keyboard.BrowserMultimediaKeysUseMSApis.....	322
Device.Input.Keyboard.CharmsKey.....	322
Device.Input.Keyboard.DynamicKeyboards.....	323
Device.Input.Keyboard.HotKeyFunctionAPI.....	323
Device.Input.Keyboard.KernelModeDriversUseWdfKmdf.....	324
Device.Input.Keyboard.LogoFlagKey.....	324
Device.Input.Keyboard.MultipleKeyboard.....	324
Device.Input.Keyboard.ScanCode.....	325
Device.Input.LampArray.....	325
Device.Input.LampArray.Accessibility.....	325
Device.Input.LampArray.AutonomousMode.....	326
Device.Input.LampArray.Brightness.....	326

Device.Input.LampArray.ColorAccuracy	327
Device.Input.LampArray.ColorLevels	327
Device.Input.LampArray.Control	327
Device.Input.LampArray.DeviceConnectionType	328
Device.Input.LampArray.DeviceFriendlyName	328
Device.Input.LampArray.DynamicLightingAPIs	329
Device.Input.LampArray.FWOverride	329
Device.Input.LampArray.HardwareID	329
Device.Input.LampArray.HIDCompliant	330
Device.Input.LampArray.LampArrayApp	330
Device.Input.LampArray.LampArrayKind	331
Device.Input.LampArray.LampArrayPurpose	331
Device.Input.LampArray.LampPositionTolerance	331
Device.Input.LampArray.LampID	332
Device.Input.LampArray.RefreshRate	332
Device.Input.LampArray.VirtualKeyMapping	333
Device.Input.Location	333
Device.Input.Location.AssistedGNSS	333
Device.Input.Location.Base	334
Device.Input.Location.Geofencing	336
Device.Input.Location.SUPL	337
Device.Input.Location.V2UPL	338
Device.Input.PointDraw	338
Device.Input.PointDraw.KernelModeDriversUseWdfKmdf	338
Device.Input.Sensor	338
Device.Input.Sensor.Accelerometer	338
Device.Input.Sensor.Accelerometer.DataFieldProperties	338
Device.Input.Sensor.Accelerometer.DataTypes	338
Device.Input.Sensor.Accelerometer.DynamicRange	339
Device.Input.Sensor.Accelerometer.MinimumReportInterval	339
Device.Input.Sensor.Accelerometer.Thresholds	339
Device.Input.Sensor.ActivitySensor	339
Device.Input.Sensor.ActivitySensor.DataTypes	339
Device.Input.Sensor.ActivitySensor.History	340
Device.Input.Sensor.ActivitySensor.MinimumReportInterval	340
Device.Input.Sensor.ActivitySensor.Thresholds	340

Device.Input.Sensor.AmbientLightSensor	341
Device.Input.Sensor.AmbientLightSensor.AmbientLightResponseCurve	341
Device.Input.Sensor.AmbientLightSensor.ColorCapable	341
Device.Input.Sensor.AmbientLightSensor.DataFieldProperties	342
Device.Input.Sensor.AmbientLightSensor.DataTypes	342
Device.Input.Sensor.AmbientLightSensor.MinimumReportInterval	342
Device.Input.Sensor.AmbientLightSensor.Thresholds	342
Device.Input.Sensor.Barometer	343
Device.Input.Sensor.Barometer.DataTypes	343
Device.Input.Sensor.Barometer.Thresholds	343
Device.Input.Sensor.CustomSensor	343
Device.Input.Sensor.CustomSensor.DataTypes	343
Device.Input.Sensor.Orientation	344
Device.Input.Sensor.Orientation.DataFieldProperties	344
Device.Input.Sensor.Orientation.DataTypes	344
Device.Input.Sensor.Orientation.Thresholds	344
Device.Input.Sensor.Orientation.MinimumReportInterval	344
Device.Input.Sensor.General	345
Device.Input.Sensor.General.DataTypes	345
Device.Input.Sensor.General.DriverFramework	345
Device.Input.Sensor.General.ReportInterval	345
Device.Input.Sensor.General.StartStop	345
Device.Input.Sensor.GeomagneticOrientation	346
Device.Input.Sensor.GeomagneticOrientationSensor.DataTypes	346
Device.Input.Sensor.GeomagneticOrientationSensor.MinimumReportInterval	346
Device.Input.Sensor.GeomagneticOrientationSensor.Thresholds	346
Device.Input.Sensor.GravityVectorSensor	346
Device.Input.Sensor.GravityVectorSensor.DataTypes	346
Device.Input.Sensor.GravityVectorSensor.Thresholds	347
Device.Input.Sensor.Gyrometer	347
Device.Input.Sensor.Gyrometer.DataFieldProperties	347
Device.Input.Sensor.Gyrometer.DataTypes	347
Device.Input.Sensor.Gyrometer.MinimumReportInterval	348
Device.Input.Sensor.Gyrometer.Thresholds	348
Device.Input.Sensor.LinearAccelerometer	348
Device.Input.Sensor.LinearAccelerometer.DataTypes	348

Device.Input.Sensor.LinearAccelerometer.Thresholds	348
Device.Input.Sensor.Magnetometer.....	349
Device.Input.Sensor.Magnetometer.DataFieldProperties.....	349
Device.Input.Sensor.Magnetometer.DataTypes	349
Device.Input.Sensor.Magnetometer.MinimumReportInterval	349
Device.Input.Sensor.Magnetometer.Thresholds.....	350
Device.Input.Sensor.Pedometer	350
Device.Input.Sensor.Pedometer.DataTypes.....	350
Device.Input.Sensor.Pedometer.History.....	350
Device.Input.Sensor.Pedometer.MinimumReportInterval	350
Device.Input.Sensor.Pedometer.Thresholds	351
Device.Input.Sensor.ProximitySensor	351
Device.Input.Sensor.ProximitySensor.DataTypes	351
Device.Input.Sensor.ProximitySensor.LongRangeProximity	351
Device.Input.Sensor.ProximitySensor.ChangeNotification.....	352
Device.Input.Sensor.HumanPresenceSensor	352
Device.Input.Sensor.HumanPresenceSensor.ApproachDetectionLatency.....	353
Device.Input.Sensor.HumanPresenceSensor.AttentionLatency	353
Device.Input.Sensor.HumanPresenceSensor.DetectionDistanceAccuracy	353
Device.Input.Sensor.HumanPresenceSensor.DistanceReporting	353
Device.Input.Sensor.HumanPresenceSensor.DynamicRange.....	354
Device.Input.Sensor.HumanPresenceSensor.ExtensionMechanismLatency	354
Device.Input.Sensor.HumanPresenceSensor.FriendlyName	354
Device.Input.Sensor.HumanPresenceSensor.InboxProximitySensor.....	354
Device.Input.Sensor.HumanPresenceSensor.IsValid	354
Device.Input.Sensor.HumanPresenceSensor.PowerConsumption	355
Device.Input.Sensor.HumanPresenceSensor.USBSelectiveSuspend	355
Device.Input.Sensor.HumanPresenceSensor.ReportedProperties	355
Device.Input.Sensor.RelativeOrientationSensor	355
Device.Input.Sensor.RelativeOrientationSensor.DataTypes.....	355
Device.Input.Sensor.RelativeOrientationSensor.MinimumReportInterval	356
Device.Input.Sensor.RelativeOrientationSensor.Thresholds	356
Device.Input.Sensor.SimpleDeviceOrientation	356
Device.Input.Sensor.SimpleDeviceOrientation.DataTypes	356
Device.Input.Sensor.SimpleDeviceOrientation.MaximumReportDelay.....	357
Device.Input.Sensor.SimpleDeviceOrientation.SimpleDeviceOrientationChangeNotification	357

Device.Input.SmartCardMiniDriver	357
Device.Input.SmartCardMiniDriver.DoNotStopWhenResourcesAreUnavailable	357
Device.Input.SmartCardMiniDriver.SpecsAndCertifications	357
Device.Input.SmartCardMiniDriver.SupportMultipleInstancesOnASystem	359
Device.Input.SmartCardReader	359
Device.Input.SmartCardReader.PinDataEntryKeyboardCompliesWithIso	359
Device.Input.SmartCardReader.SmartCardService	359
Device.Input.SmartCardReader.Supports258And259BytePackets	360
Device.Input.SmartCardReader.SupportsDirectAndInverseConvention	360
Device.Input.SmartCardReader.SupportsInsertionAndRemovalMonitor	360
Device.Input.SmartCardReader.SupportsMinClockFrequency	361
Device.Input.SmartCardReader.SupportsMinDataRateOf9600bps	361
Device.Input.SmartCardReader.SupportsNegotiableAndSpecificModes	361
Device.Input.SmartCardReader.SupportsResetCommand	361
Device.Input.SmartCardReader.UsbCcidCompliesWithUsbDeviceClassSpec	362
Device.Input.SmartCardReader.UsbCcidIssuesNak	362
Device.Network.DevFund	363
Device.Network.DevFund.NdisVersion	363
Device.Network.DevFund.NPOS	363
Device.Network.DevFund.NetAdapterCxVersion	363
Device.Network.DevFund.SelectiveSuspend	364
Device.Network.LAN.Base	364
Device.Network.LAN.Base.100MbOrGreater	364
Device.Network.LAN.Base.32MulticastAddresses	364
Device.Network.LAN.Base.AdvProperties	364
Device.Network.LAN.Base.AnyBoundary	365
Device.Network.LAN.Base.IPv4AndIPv6OffloadParity	365
Device.Network.LAN.Base.NDISCalls	365
Device.Network.LAN.Base.NDISRequirements	366
Device.Network.LAN.Base.NetAdapterCxCalls	366
Device.Network.LAN.Base.NetAdapterCxRequirements	366
Device.Network.LAN.Base.PacketFiltering	366
Device.Network.LAN.Base.PreserveOSServices	367
Device.Network.LAN.Base.PriorityVLAN	367
Device.Network.LAN.Base.ShortPacketPadding	367
Device.Network.LAN.Base.SupportIEEE8023	367

Device.Network.LAN.ChecksumOffload	368
Device.Network.LAN.ChecksumOffload.ChecksumOffload	368
Device.Network.LAN.CS	368
Device.Network.LAN.CS.NetAdapterCx	368
Device.Network.LAN.CS.NetworkWake	369
Device.Network.LAN.CS.PresenceOffload	369
Device.Network.LAN.CS.ReliableCSConnectivity	370
Device.Network.LAN.CS.WakeEvents	370
Device.Network.LAN.CS.WakeReasonDetection	371
Device.Network.LAN.DCB	371
Device.Network.LAN.DCB.DCB	371
Device.Network.LAN.GRE	371
Device.Network.LAN.GRE.GREPacketTaskOffloads	371
Device.Network.LAN.IPsec	372
Device.Network.LAN.IPsec.IPsec	372
Device.Network.LAN.KRDMA	372
Device.Network.LAN.KRDMA.KRDMA	373
Device.Network.LAN.LargeSendOffload	373
Device.Network.LAN.LargeSendOffload.LargeSendOffload	373
Device.Network.LAN.MSI	374
Device.Network.LAN.MTUSize	374
Device.Network.LAN.MTUSize	374
Device.Network.LAN.MTUSize.EncapOverhead	374
Device.Network.LAN.NDISPollMode	374
Device.Network.LAN.NDISPollMode	374
Device.Network.LAN.PM	375
Device.Network.LAN.PM.PowMgmtNDIS	375
Device.Network.LAN.PM.WakeOnLANPatterns	375
Device.Network.LAN.PM.WakePacket	376
Device.Network.LAN.QoSOffload	376
Device.Network.LAN.QoSOffload.QoSOffload	376
Device.Network.LAN.RSC	376
Device.Network.LAN.RSC.RSC	377
Device.Network.LAN.RSS	377
Device.Network.LAN.RSS.RSS	377
Device.Network.LAN.RSS.SetHashFunctionTypeAndValue	378

Device.Network.LAN.RSS.SupportIndirectionTablesSizes.....	378
Device.Network.LAN.RSS.SupportToeplitzHashFunction	379
Device.Network.LAN.RSS.SupportUpdatesToRSSInfo	379
Device.Network.LAN.SDDC.Premium.....	379
Device.Network.LAN.SDDC.Premium.BasicFunction.....	380
Device.Network.LAN.SDDC.Premium.CloudStress.....	381
Device.Network.LAN.SDDC.Standard.....	381
Device.Network.LAN.SDDC.Standard.Functions.....	381
Device.Network.LAN.SDDC.Standard.CloudStress	383
Device.Network.LAN.SRIOV	383
Device.Network.LAN.SRIOV.SRIOV.....	383
Device.Network.LAN.SRIOV.VF	384
Device.Network.LAN.SRIOV.VF.VF	384
Device.Network.LAN.SRIOV.VF.NDKMode3	384
Device.Network.LAN.Timestamp	385
Device.Network.LAN.Timestamp.Timestamp.....	385
Device.Network.LAN.UDP	385
Device.Network.LAN.UDP.SegmentationOffload	385
Device.Network.LAN.VMMQ	385
Device.Network.LAN.VMMQ.VirtualMachineMultipleQueues.....	385
Device.Network.LAN.VMMQ.RSSv2	386
Device.Network.LAN.VMQ.....	386
Device.Network.LAN.VMQ.VirtualMachineQueues	386
Device.Network.LAN.VXLAN	387
Device.Network.LAN.VXLAN.VXLANPacketTaskOffloads	387
Device.Network.MobileBroadband.CDMA(All CDMA test cases are optional – they are not required for certification)	387
Device.Network.MobileBroadband.CDMA.Discretionary	387
Device.Network.MobileBroadband.ComplyWithBaseReq	388
Device.Network.MobileBroadband.NRNSA	391
Device.Network.MobileBroadband.NRNSA.Discretionary	391
Device.Network.MobileBroadband.NRSA	392
Device.Network.MobileBroadband.NRSA.Discretionary	392
Device.Network.MobileBroadband.BaseStationInfo.Discretionary.....	394
Device.Network.MobileBroadband.UiccExtensions	395
Device.Network.MobileBroadband.UiccExtensions.LowLevelUiccAccess	395

Device.Network.MobileBroadband.UiccExtensions.AppAccess	396
Device.Network.MobileBroadband.MultiCarrierFunctionality	397
Device.Network.MobileBroadband.MultiplePDPCContext	397
Device.Network.MobileBroadband.SAR	399
Device.Network.MobileBroadband.SAR.Discretionary	399
Device.Network.MobileBroadband.ImplementSMS	400
Device.Network.MobileBroadband.USSD.....	401
Device.Network.Switch.Manageability	402
Device.Network.Switch.Manageability.NetworkSwitchProfile	402
Device.Network.Switch.Manageability.NetworkSwitchProfileView	402
Device.Network.WLAN.....	403
Device.Network.WLAN.SupportConnectionToAP	404
Device.Network.WLAN.SupportConnectionToAP.ConnectionToAP (Mandatory).....	404
Device.Network.WLAN.SupportConnectionToWiFiAP	411
Device.Network.WLAN.SupportConnectionToWiFiAP.ConnectionToWiFiAP (Mandatory for all Wi-Fi Radios other than 60GHz)	411
Device.Network.WLAN.SupportConnectionToWiGigAP	415
Device.Network.WLAN.SupportConnectionToWiGigAP.ConnectionToWiGigAP (Mandatory for 60GHz Radio's).....	415
Device.Network.WLAN.SupportWakeFromLowPower	416
Device.Network.WLAN.SupportWakeFromLowPower.WakeFromLowPower.....	416
Device.Network.WLAN.SupportMACAddressRandomization	417
Device.Network.WLAN.SupportMACAddressRandomization.MACAddressRandomization (WDI drivers only).....	417
Device.Network.WLAN.SupportHotspot2Dot0.....	417
Device.Network.WLAN.SupportHotspot2Dot0.Hotspot2Dot0 (WDI drivers only)	417
Device.Network.WLAN.SupportDot11W	417
Device.Network.WLAN.SupportDot11W.Dot11W	417
Device.Network.WLAN.SupportFIPS.....	418
Device.Network.WLAN.SupportFIPS.FIPS	418
Device.Network.WLAN.SupportWiFiDirect	418
Device.Network.WLAN.SupportWiFiDirect.WiFiDirect	418
Device.Network.WLAN.SupportWiFiDirectConcurrency	421
Device.Network.WLAN.SupportWiFiDirectConcurrency.WiFiDirectConcurrency	421
Device.Network.WLAN.SupportWiFiDirectServices	426
Device.Network.WLAN.SupportWiFiDirectServices.WiFiDirectServices (WDI drivers only).....	426
Device.Network.WLAN.SupportWirelessDocking	427
Device.Network.WLAN.SupportWirelessDocking (WDI drivers only)	427

Device.Network.WLAN.SupportHostedNetwork.....	427
Device.Network.WLAN.SupportHostedNetwork.HostedNetwork (Native Wi-Fi drivers only).....	427
Device.Portable.Core.....	428
Device.Portable.Core.AudioCodec	428
Device.Portable.Core.CustomDeviceServices.....	429
Device.Portable.Core.DeviceServices.....	430
Device.Portable.Core.MediaSync	432
Device.Portable.Core.ModelID	433
Device.Portable.Core.MTP.....	433
Device.Portable.Core.MTPFunctionality	438
Device.Portable.Core.MTPMultiSession	438
Device.Portable.Core.MTPObjectProperties	439
Device.Portable.Core.MTPStreams	441
Device.Portable.Core.TransportBluetooth.....	442
Device.Portable.Core.TransportIP.....	442
Device.Portable.Core.TransportIPDLNA	444
Device.Portable.Core.TransportUSB.....	444
Device.Portable.Core.VideoCodec.....	445
Device.Portable.DigitalCamera	447
Device.Portable.DigitalCamera.MTP	447
Device.Portable.DigitalVideoCamera	450
Device.Portable.DigitalVideoCamera.MTP	450
Device.Portable.MediaPlayer	453
Device.Portable.MediaPlayer.MTP	453
Device.Portable.MobilePhone.....	456
Device.Portable.MobilePhone.MTP.....	456
Device.Stprage	459
Device.Storage.Controller	459
Device.Storage.Controller.BasicFunction	459
Device.Storage.Controller.ClassCode	460
Device.Storage.Controller.InfFile	460
Device.Storage.Controller.MiniportDriverModel.....	461
Device.Storage.Controller.Ata	462
Device.Storage.Controller.Ata.Interface	462
Device.Storage.Controller.Boot.BasicFunction	462
Device.Storage.Controller.Boot.BitLocker	462

Device.Storage.Controller.Fc	463
Device.Storage.Controller.Fc.Interface	463
Device.Storage.Controller.Flush	463
Device.Storage.Controller.Flush.BasicFunction	463
Device.Storage.Controller.Iscsi	464
Device.Storage.Controller.Iscsi.Interface	464
Device.Storage.Controller.Optical	465
Device.Storage.Controller.Optical.BasicFunction	465
Device.Storage.Controller.PassThroughSupport	465
Device.Storage.Controller.PassThroughSupport.BasicFunction	465
Device.Storage.Controller.Raid	466
Device.Storage.Controller.Raid.BasicFunction	466
Device.Storage.Controller.Sas	466
Device.Storage.Controller.Sas.Interface	467
Device.Storage.Controller.Sata	467
Device.Storage.Controller.Sata.Interface	467
Device.Storage.Controller.SD	468
Device.Storage.Controller.SD.BasicFunction	468
Device.Storage.ControllerDrive.NVMe	468
Device.Storage.ControllerDrive.NVMe.BasicFunction	468
Device.Storage.Hd	471
Device.Storage.Hd.BasicFunction	471
Device.Storage.Hd.PhysicalSectorSizeReportsAccurately	471
Device.Storage.Hd.RotationalRate	472
Device.Storage.Hd.1394	473
Device.Storage.Hd.1394.Compliance	473
Device.Storage.Hd.Ata	473
Device.Storage.Hd.Ata.BasicFunction	473
Device.Storage.Hd.Ata.Dma	473
Device.Storage.Hd.AtaProtocol	474
Device.Storage.Hd.AtaProtocol.Performance	474
Device.Storage.Hd.AtaProtocol.Protocol	474
Device.Storage.Hd.DataVerification	475
Device.Storage.Hd.DataVerification.BasicFunction	475
Device.Storage.Hd.Ehdd	475
Device.Storage.Hd.Ehdd.Compliance	475

Device.Storage.Hd.EMMC.....	477
Device.Storage.Hd.EMMC.BasicFunction.....	477
Device.Storage.Hd.EnhancedStorage.....	478
Device.Storage.Hd.EnhancedStorage.1667Compliance.....	478
Device.Storage.Hd.FibreChannel.....	479
Device.Storage.Hd.FibreChannel.Compliance.....	479
Device.Storage.Hd.Flush.....	479
Device.Storage.Hd.Flush.BasicFunction.....	479
Device.Storage.Hd.Iscsi.....	479
Device.Storage.Hd.Iscsi.BasicFunction.....	479
Device.Storage.Hd.MultipleAccess.PersistentReservation.....	480
Device.Storage.Hd.MultipleAccess.PersistentReservation.BasicFunction.....	481
Device.Storage.Hd.RaidArray.....	481
Device.Storage.Hd.RaidArray.BasicFunction.....	481
Device.Storage.Hd.ReadZeroOnTrimUnmap.....	482
Device.Storage.Hd.ReadZeroOnTrimUnmap.BasicFunction.....	482
Device.Storage.Hd.RemovableMedia.....	482
Device.Storage.Hd.RemovableMedia.BasicFunction.....	482
Device.Storage.Hd.Sas.....	482
Device.Storage.Hd.Sas.ComplyWithIndustrySpec.....	482
Device.Storage.Hd.Sata.....	484
Device.Storage.Hd.Sata.BasicFunction.....	484
Device.Storage.Hd.Sata.HybridInformation.....	485
Device.Storage.Hd.Sata.HybridInformation.BasicFunction.....	485
Device.Storage.Hd.Scsi.....	489
Device.Storage.Hd.Scsi.Connectors.....	489
Device.Storage.Hd.Scsi.ParallelInterface.....	489
Device.Storage.Hd.Scsi.ReliabilityCounters.....	490
Device.Storage.Hd.Scsi.ReliabilityCounters.BasicFunction.....	490
Device.Storage.Hd.ScsiProtocol.....	491
Device.Storage.Hd.ScsiProtocol.ReferenceSpec.....	491
Device.Storage.Hd.ScsiProtocol.SamCompliance.....	491
Device.Storage.Hd.ScsiProtocol.SpcCompliance.....	492
Device.Storage.Hd.SMART.....	494
Device.Storage.Hd.SMART.HealthWarning.....	494
Device.Storage.Hd.Trim.....	495

Device.Storage.Hd.Trim.BasicFunction	495
Device.Storage.Hd.Uas	495
Device.Storage.Hd.Uas.Compliance	495
Device.Storage.Hd.UasOnEHCI	496
Device.Storage.Hd.UasOnEHCI.BasicFunction	496
Device.Storage.Hd.Usb	496
Device.Storage.Hd.Usb.Compatibility	496
Device.Storage.Hd.Usb3	497
Device.Storage.Hd.Usb3.Compliance	497
Device.Storage.Hd.WindowsToGoCapableUSBDrive	498
Device.Storage.Hd.WindowsToGoCapableUSBDrive.WindowsToGoCapableUSBDrive	498
Device.Storage.Optical	499
Device.Storage.Optical.CdRawRecording	499
Device.Storage.Optical.CommandPerformance	499
Device.Storage.Optical.DriveDefinition	502
Device.Storage.Optical.Features	502
Device.Storage.Optical.MmcVersion	502
Device.Storage.Optical.Profiles	503
Device.Storage.Optical.RealTimeStreaming	503
Device.Storage.Optical.BluRayReader	503
Device.Storage.Optical.BluRayReader.Profiles	503
Device.Storage.Optical.BluRayWriter	504
Device.Storage.Optical.BluRayWriter.Profiles	504
Device.Storage.Optical.Sata	504
Device.Storage.Optical.Sata.AsynchronousNotification	504
Device.Streaming.Camera.Base	504
Device.Streaming.Camera.Base.AVStreamClassInterfaceAndWDM	504
Device.Streaming.Camera.Base.MediaFoundation	505
Device.Streaming.Camera.Base.MediaTypes	505
Device.Streaming.Camera.Base.MultipleClientAppSupport	506
Device.Streaming.Camera.Base.PnPAndPowerPolicies	506
Device.Streaming.Camera.Base.SurpriseRemoval	506
Device.Streaming.Camera.Sensor.KsCategoryRegistration	506
Device.Streaming.Camera.Base.UsageIndicator	507
Device.Streaming.Camera.VideoCapture.DirectShow	507
Device.Streaming.Camera.VideoCapture.KSCategoryRegistration	507

Device.Streaming.Camera.Base.Sharing	507
Device.Streaming.Camera.Sensor.FaceAuthMode	507
Device.Streaming.Camera.Base.Sharing.CustomMediaSource	508
Device.Streaming.Camera.Base.Sharing.DeviceMFT	508
Device.Streaming.Camera.Base.Sharing.PlatformDeviceMFT	508
Device.Streaming.Camera.Base.Sharing.MemoryAllocation	509
Device.Streaming.Camera.Base.Sharing.NonConsumableData	509
Device.Streaming.Camera.Base.HardwarePrivacy	509
Device.Streaming.Camera.Base.HardwarePrivacy	509
Device.Streaming.Camera.Sharing.VirtualCamera	510
Device.Streaming.Camera.Sharing.CameraAIEffects	510
Device.Streaming.Camera.UVC	511
Device.Streaming.Camera.UVC.USBClassDriver	511
Device.Streaming.Camera.UVC.UVCCacheDeprecation	511
Device.Streaming.Camera.UVC.UVCDriver	511
Device.Streaming	512
Device.Streaming.D3D9.H264	512
Device.Streaming.D3D9.H264.Playback	512
Device.Streaming.D3D9.MPEG2	512
Device.Streaming.D3D9.MPEG2.Playback	512
Device.Streaming.D3D9.VC1	512
Device.Streaming.D3D9.VC1.Playback	512
Device.Streaming.D3D11.HDRStreaming	513
Device.Streaming.D3D11.HDRStreaming.VP9	513
Device.Streaming.D3D11.HDRStreaming.HEVC	513
Device.Streaming.D3D11.AV1.Profile0.4k.Playback	513
Device.Streaming.D3D11.H264	513
Device.Streaming.D3D11.H264.4k.Playback	513
Device.Streaming.D3D11.HDRStreaming	514
Device.Streaming.D3D11.HDRStreaming.HEVC.Playback	514
Device.Streaming.D3D11.HDRStreaming.VP9.Playback	514
Device.Streaming.D3D11.HEVC	514
Device.Streaming.D3D11.HEVC.Main.4k.Playback	514
Device.Streaming.D3D11.HEVC.Main.Playback	515
Device.Streaming.D3D11.HEVC.Main10.4k.Playback	515
Device.Streaming.D3D11.HEVC.Main10.Playback	515

Device.Streaming.D3D11.VP8	515
Device.Streaming.D3D11.VP8.Playback	515
Device.Streaming.D3D11.VP9	515
Device.Streaming.D3D11.VP9.Profile0.4k.Playback	515
Device.Streaming.D3D11.VP9.Profile0.Playback	516
Device.Streaming.D3D11.VP9.Profile2.Playback	516
Device.Streaming.D3D12.H264.Playback	516
Device.Streaming.D3D12.HEVC	516
Device.Streaming.D3D12.HEVC.Main.4k.Playback	516
Device.Streaming.D3D12.HEVC.Main.Playback	517
Device.Streaming.D3D12.HEVC.Main10.4k.Playback	517
Device.Streaming.D3D12.AV1.Profile0	518
Device.Streaming.D3D12.AV1.Profile0.Playback	518
Device.Streaming.D3D12.AV1.Profile0.4k.Playback	518
Device.Streaming.HMFT	518
Device.Streaming.HMFT.Decoding	518
Device.Streaming.HMFT.Encoding	520
Device.Streaming.HMFT.H264.Encoding	523
Device.Streaming.HMFT.H265.Encoding	530
Device.Streaming.HMFT.VP8.Encoding	537
Device.Streaming.HMFT.VP9.Encoding	543
Device.Streaming.HMFT.AV1	548
Device.Streaming.HMFT.AV1.Encoding	548
Device.Streaming.HMFT.Displayable.MJPEG	549
Device.Streaming.HMFT.Displyable.MJPEG.Decoding	549
Device.Streaming.HMFT.H263	549
Device.Streaming.HMFT.H263.Decoding	549
Device.Streaming.HMFT.HEVC	549
Device.Streaming.HMFT.HEVC.Encoding	549
Device.Streaming.HMFT.HEVC.MAIN10.Encoding	550
Device.Streaming.HMFT.MJPEG	550
Device.Streaming.HMFT.MJPEG.Decoding	550
Device.Streaming.HMFT.MPEG4	550
Device.Streaming.HMFT.MPEG4.Decoding	550
Device.Streaming.HWDRM.H264	550
Device.Streaming.HWDRM.H264.CBCS.Playback	550

Device.Streaming.HWDRM.H264.CENC.Playback	551
Device.Streaming.HWDRM.H264.Playback	551
Device.Streaming.HWDRM.HEVC	551
Device.Streaming.HWDRM.HEVC.Main.CENC.Playback	551
Device.Streaming.HWDRM.HEVC.Main.Playback	551
Device.Streaming.HWDRM.HEVC.Main10.Playback	552
Device.Streaming.HWDRM.VP9	552
Device.Streaming.HWDRM.VP9.Profile2.10bit.CENC.Playback	552
Device.Streaming.HWDRM.AV1.Profile0	553
Device.Streaming.HWDRM.AV1.Profile0.Playback	553
Device.Streaming.HWDRM.AV1.Profile0.CBCS.Playback	553
Device.Streaming.HWDRM.AV1.Profile0.CENC.Playback	553
Device.TrustedPlatformModule	553
Device.TrustedPlatformModule.TPM20	553
Device.TrustedPlatformModule.TPM20.Features	554
Device.TrustedPlatformModule.TPM20.MinimumPerformance	556
Device.TrustedPlatformModule.TPM20.EKCert	560

Glossary

If-Implemented: An If-Implemented requirement is a mandatory requirement for an optional feature or function. The product is not required to implement the feature or function defined by the requirement, however if the feature or function is implemented, it must comply with the requirement.

For example: Systems are not required to support USB Type C in the current requirements. If a system does support USB Type C functionality, then the system is required to comply with all of the defined USB Type C requirements.

Optional: An optional requirement is a recommendation that Microsoft strongly encourages be implemented when a product supports the described features or functions. This optional requirement is not enforced during the Windows Hardware Compatibility Program qualification. Therefore, it is possible to become Windows Hardware Compatibility Program qualified and not comply with the given optional requirement.

For example: Microsoft recommends that HDD and SSD storage devices support Self-Monitoring, Analysis and Reporting Technology (SMART) metrics per the ATA8-ACS4 specification in order to reduce the chance of unexpected device failure or data loss. A storage device is not required to comply with this optional requirement, even when the device supports SMART.

Device Requirements

Device.Audio.APO

This APO must match all APO tests.

Device.Audio.APO.BestPractices

APOs should follow best practices for Windows APOs

Applies to	Windows 11 Client x64

Description

All APOs (Audio Processing Objects) should follow the best practices outlined in the Windows APO Best Practices white paper.

This white paper can be found on MSDN: <http://go.microsoft.com/fwlink/?LinkId=716913>

Device.Audio.APO.ClassRegistration

Validate Audio Processing Objects class registration

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

INFs that contain an AddReg section that targets the registry key AudioEngine\AudioProcessingObjects must have class AudioProcessingObject.

Device.Audio.APO.MicArrayRawData

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 recommended for onboard fixed-position microphone array (multiple combined elements) on a system to follow "Speech Platform Device Recommendations Specification" for an optimized experience.

Device.Audio.APO.RelativeRegistration

Validate Audio Processing Objects relative registration

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

INFs that register Audio Processing Objects should target HKR registry keys, not HKCR.

Device.Audio.APO.ThirdParty

APOs should follow best practices for Windows APOs

Applies to	Windows 11 Client x64

Description

All APOs (Audio Processing Objects) shall be tested for basic functionality to ensure that APO crashes won't negatively affect user experiences with audio. The basic functionality required by the APOs covers a variety of areas. Basic functionality is required by both APOs associated with specific hardware and APOs distributed as an effect package (a.k.a. effect pack). These areas include the following.

- APO initialization is tested to ensure that APOs will initialize properly on Windows systems, since there have been issues related to this in the past.
- The APO must be able to be activated and deactivated immediately to simulate quick connect and disconnect scenarios.
- A frame count validation test is done to make sure that bad frame counts won't negatively affect the quality of audio on the system.
- A basic data streaming test is done to ensure that data can be streamed successfully and that invalid values of data such as infinite or NaN are not being returned from the APO.
- If an APO has a custom format, a test is done to make sure that custom format is a valid format for use on a Windows system.
- APOs that implement AEC, and require loopback reference streams shall not use WASAPI to obtain the reference streams. They shall be using the new interfaces provided for this purpose instead.
- APOs will be stress tested to ensure that they can handle normal heavy use conditions that will likely occur on a system.
- APOs shall not call any of the newly disallowed APIs such as ones to terminate thread.
- APOs shall use the new settings framework for APOs.
- APOs shall not use the register and unregister APIs provided by WASAPI.
- APOs shall not use createthread for realtime processing.
- APOs shall not create any logging files on disk.
- APOs shall not use WASAPI to obtain references to streams for things such as AEC using a loopback reference.
- APOs are validated across upgrade against the above criteria in order to catch blue screen issues, APO hangs impacting Windows Update, and loss-of-audio issues after upgrade.
- APOs shall return only the audio effects that are turned on to the list of controllable audio effects from IAudioSystemEffects3 instead of the entire list of effects.
- APOs shall not link to particular audio parts for the purposes of something like Mic boosting.
- APOs shall not use custom notifications to listen for any type of volume.
- If an effect pack is installed for a specific type of endpoint, HLK tests will require the presence of that type of endpoint so that tests can be run against that endpoint. If the type of endpoint is not present, the HLK tests will fail.
- A test is done to verify the effect pack extension INF.

- An effect pack associated with an APO will be tested with all the criteria within Device.Audio.APO.ThirdParty to determine if everything will work as expected for the APO.
- A test is done to ensure the instantiation of multiple AEC APOs per endpoint using the SetEchoCancellationRenderEndpoint API. Multiple AEC APOs should be able to do echo cancellation concurrently per endpoint.

In Windows 11, new APIs were introduced to provide more reliable interaction between an APO, the OS, and applications. More detailed information about the new interfaces (Acoustic Echo Cancellation, Settings, Notifications, Logging, Threading, APO discovery and control for applications) can be found at:

<https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/windows-11-apis-for-audio-processing-objects>

Device.Audio.APO.VoiceClarity.Supported

Audio capture pipeline with AEC (acoustic echo cancelation) and other voice quality features.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Voice Clarity is a standalone MFX APO provided by Microsoft. Its enablement follows the OEM APO deployment model (see: <https://learn.microsoft.com/en-us/windows-hardware/drivers/audio/implementing-audio-processing-objects>). Voice Clarity is not an in-box feature of Windows, nor is it an Effect Pack. Voice Clarity must not be chained with other non-linear audio processing effects, either before or after it. Enablement of Voice Clarity is optional. These requirements only apply if the Voice Clarity extension package is implemented.

Design Criteria

1. Clocking
The device must have a single clock domain for all built-in microphones and speakers (no relative drift by design).
2. No Pipeline Chaining
Pipelines (hardware, firmware, or software) implementing echo control, noise suppression, beamforming, automatic gain control, compression, or any other nonlinear effect must not be inserted before Voice Clarity. Applications must use effects discovery API to choose between offloading processing, or capturing a raw stream and doing their own processing.
3. Speaker Loopback Sampling Location (HLKTestLoopbackSample)
Speaker loopback signal must be sampled before volume control.
4. Microphone Noise Floor Matching
The difference of noise floor powers of all microphones on the device must stay within 3dB range. This must be satisfied for each of the following:
 - a. Noise floor consists of microphone self-noise (no ambient noise).
 - b. Noise floor consists of microphone self-noise + diffuse ambient noise.
 - c. Noise floor consists of microphone self-noise + diffuse ambient noise + device self-noise (e.g., high-speed fan).
5. No Microphone Boost (HLKTestMicBoost)
A boost subunit must not be used.

6. Speaker to Microphone Coupling (HLKTestCouplingFactor)
Measured speaker to microphone coupling must be lower than -10dB.

Performance Criteria

1. Speaker Loopback Bandwidth (HLKTestLoopbackBandwidth)
Loopback signal must not be bandlimited relative to the rendered signal.
2. Speaker Loopback to Microphone Causality and Latency (HLKTestCausality)
QPC timestamps and buffers must be consistent with speaker signal being produced before the corresponding measurement at the microphones.
Latency from speaker stimulus (default mode) to microphone capture (raw mode) must be between 0 ms and 40 ms.
3. Microphone Signal Resolution and Dynamic Range (HLKTestMicSaturation, HLKTestMicDynamicRange)
When the microphone volume is set to 0 dB gain, the microphone signal must be guaranteed to:
 - a. not saturate at full render volume from built-in speakers. (HLKTestMicSaturation)
 - b. represent the full dynamic range of the microphones, with no loss of significant bits. (HLKTestMicDynamicRange)
4. Driver Render and Capture Time Stamps (HLKTestDriverTimeStamp)
These requirements must be met on all power modes where Voice Clarity can be used:
 - a. Timestamp jitter must be < 4ms.
 - b. Render to capture relative delay jitter must be < 10ms.

Device Properties

1. If the device has more than one front-facing microphone, the microphone array geometry must be provided and match hardware geometry specified in *PKEY_Devices_MicrophoneArray_Geometry*.
 - a. If this property is unspecified, Voice Clarity performs monophonic processing using only the first microphone.
2. Microphone sensitivity must be provided and match sensitivity with 0dB device gain. Specified in *PKEY_Devices_AudioDevice_Microphone_SensitivityInDbfs2*.
3. Microphone SNR must be provided and be consistent with in-system SNR of the microphone (without processing). Specified in *PKEY_Devices_AudioDevice_Microphone_SignalToNoiseRatioInDb*.

Device.Audio.Base

This device must match all base tests.

Device.Audio.Base.AudioProcessing

Audio devices must support proper audio processing discovery and control.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Drivers

At a minimum, a driver must support a raw mode or a default mode pin on any render pins. At a minimum, drivers other than Bluetooth HFP, must support raw mode on any capture pins. Bluetooth HFP may support raw or default mode on capture. 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.

Device.Audio.Base.Endpoints

Audio subsystem properly reflects current system configuration

Description

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 switching from an integrated GPU to a discrete GPU or vice versa

Exposing discrete endpoints to the OS

This subsection requirement will become effective from the Windows release after February 2019

By default, the audio driver and device must not perform hidden stream redirection, routing, switching, splitting, mixing, muxing to other exposed or hidden logical audio devices, applications or other entities and 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. This is an extension of a previous Windows v.10.0 Mobile requirement that is now applicable to all SKUs.

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 endpoints. There are two options to achieve this:

- The driver must implement the following properties for jack presence detection
 - KSPROPSETID_Jack
 - KSPROPERTY_JACK_DESCRIPTION - KSJACK_DESCRIPTION.IsConnected communicates the current connection state
 - KSPROPERTY_JACK_DESCRIPTION2 - KSJACK_DESCRIPTION2.JackCapabilities must indicate JACKDESC2_PRESENCE_DETECT_CAPABILITY
 - KSEVENTSETID_PinCapsChange
 - KSEVENT_PINCAPS_JACKINFOCHANGE - an event is always sent if KSJACK_DESCRIPTION.IsConnected changes while the KS filter interfaces are active
- The driver changes the KS filter interface state and only communicates jack description information
 - KSPROPSETID_Jack
 - KSPROPERTY_JACK_DESCRIPTION - KSJACK_DESCRIPTION.IsConnected is always set to TRUE and must never change while the interfaces are active/registered
 - KSPROPERTY_JACK_DESCRIPTION2 - is either not implemented or is implemented and KSJACK_DESCRIPTION2.JackCapabilities does not report JACKDESC2_PRESENCE_DETECT_CAPABILITY

- For AvStream drivers: call `IoSetDeviceInterfaceState` with `TRUE/FALSE` to enable/disable the interface
- For Port Class drivers: call `PcRegisterSubdevice/PcUnregisterSubdevice` to enable/disable the interface

In all cases `KSPROPERTY_JACK_DESCRIPTION` must be implemented. For devices which do not possess an audio jack, such as built-in speakers, the `KSJACK_DESCRIPTION` fields which are not applicable can be filled in with their corresponding unknown or not applicable value (i.e., `eConnTypeUnknown`, `eGeoLocNotApplicable`, `ePortConnUnknown`).

For both `KSPROPERTY_JACK_DESCRIPTION 1` and `2`, the driver needs to return one `KSMULTIPLE_ITEM` structure followed by at least one `KSJACK_DESCRIPTION` (or `KSJACK_DESCRIPTION2`) structure.

Whenever the driver changes the `KSJACK_DESCRIPTION.IsConnected` member, the driver must generate an `KSEVENT_PINCAPS_JACKINFOCHANGE` event. The `KSJACK_DESCRIPTION.IsConnected` field may be changed while the KS filters are in the disabled state without the need to also set the `KSEVENT_PINCAPS_JACKINFOCHANGE` event and the updated value will be read when the interfaces are activated.

Implementing both jack presence detection along with changing KS filter interface state is not recommended, as changing both at the same time can result in unnecessary processing which can degrade the user experience. Careful planning and validation are necessary to ensure smooth transitions between the endpoint states whenever mixing these strategies.

NOTE: Changing the filter state will result in the audio endpoint disappearing from some UI, as the endpoint will be reported as `DEVICE_STATE_NOTPRESENT` rather than `DEVICE_STATE_UNPLUGGED`. Additionally, more processing is required to transition an endpoint from `DEVICE_STATE_NOTPRESENT` to `DEVICE_STATE_ACTIVE` than is required to transition from `DEVICE_STATE_UNPLUGGED` to `DEVICE_STATE_ACTIVE`. Option a), jack presence detection, provides a better user experience for switching audio from one endpoint to another quickly and is the recommended strategy whenever possible.

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://go.microsoft.com/fwlink/?LinkId=716908>.

Device.Audio.Base.HardwareArchitecture

Audio subsystems must use a technology compatible with Windows.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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://go.microsoft.com/fwlink/?LinkId=716909](http://go.microsoft.com/fwlink/?LinkId=716909)

Device.Audio.Base.KSTopologyCompliance

Audio Device Driver provides kernel streaming topology according to the documentation in the Microsoft Windows Driver Kit.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Some important examples, not inclusive of all the driver must adhere to, are shown below. See the WDK for full disclosure.

Check Pin KsDataRange: If a datarange structure (KSDATARANGE structure) has a "Specifier" value of KSDATAFORMAT_SPECIFIER_WAVEFORMATEX, then "FormatSize" must be sizeof(KSDATARANGE_AUDIO).

KSDATARANGE_AUDIO values must be constrained as follows:

- "SampleFrequency" is between 1 Hz and 2,000,000 Hz,
- "BitsPerSample" is between 8 and 32 bits.

Check Orphaned Pins: All pins must have at least one internal connection and none can be orphaned.

Node Verifications: All nodes' pin I/O count for all node types are specified on the Microsoft Docs site. The following is a list of the required number of input and output connections for some common node types:

KS Node	Number of Inputs	Number of Outputs
KSNODETYPE_MUX	1	1
KSNODETYPE_SUM	> 1	1
KSNODETYPE_DEMUX	1	> 1
KSNODETYPE_ACOUSTIC_ECHO_CANCEL	2	2
KSNODETYPE_DEV_SPECIFIC	Not Specified	Not Specified
All other nodes	1	1

In the special case of KSNODETYPE_MUX nodes, the number of inputs can be one, but it is recommended that all nodes have at least two inputs for performance gains.

Check Orphaned Nodes: All nodes must have connections to other nodes and no orphaned nodes should be left.

Channel properties: All channel properties, including channels returning Boolean values, such as mute, must include one MembersHeader and the MembersCount field must accurately describe the number of channels. This requirement also applies to mono controls.

Device.Audio.Base.PowerManagement

Audio device must comply with related power management specifications.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

Hardware Keyword Spotter Enabled Devices

Devices that support Windows Assistants or Multiple Voice Assistant must make sure that the system enters HWDIPS on paused audio streams.

Relevant tests:

- Modern Standby Voice Activation Power Test on DC-power Source

- Modern Standby Voice Activation Power Test on AC-power Source

Reference(s)

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

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

MSDN: <http://go.microsoft.com/fwlink/?LinkId=716910>

Device.Audio.Base.SamplePositionAccuracy

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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

- $|\text{bias}| \leq 1\text{ms}$
- $|\text{skew}| \leq 1\%$
- $\text{Jitter} \leq 1\text{ms}$

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 11 Client x64 Windows 11 Client ARM64

Description

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.

Loopback pins should match the format of their corresponding default mode host render pins.

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.

At a minimum, offloaded pins must support both 48 kHz and 44.1 kHz sampling rates.

Reference(s)

MSDN: <http://go.microsoft.com/fwlink/?LinkId=716911>

MSDN: [https://msdn.microsoft.com/en-us/library/windows/hardware/ff537083\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/ff537083(v=vs.85).aspx)

Device.Audio.Base.VolumeControl

Audio driver volume controls are linear and have adequate resolution.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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 10 dB 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.

Device.Audio.Base.WaveRTConformance

Audio device is designed to be WaveRT-port-friendly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Requirements Device.Audio.Base.IMiniportWaveRTStreamNotification and Device.Audio.Base.WaveRTImplementation have been merged with this one.

A UAA HD Audio-compatible implementation meets this requirement automatically. To be considered WaveRT port friendly, the audio subsystem must support the following:

- Cyclic DMA engine with a scatter-gather list.
- Ability to split samples between pages.
- Ability to loop on buffers without software intervention.

Integrated or discrete audio device driver must be based on the Microsoft Windows WaveRT miniport WDM driver model. Requirement details are defined in the white paper titled "A Wave Port Driver for Real-Time Audio Streaming."

WaveRT drivers must support event driven audio streaming technology by implementing the IMiniportWaveRTStreamNotification interface. By supporting this functionality in the driver, future Microsoft Windows operating systems will be able to employ more efficient techniques for supplying and retrieving data buffers from the audio device. In turn, this will reduce the overall audio latency on the system.

The legacy ports WaveCyclic or WavePCI are not used to support audio devices on Windows.

Design Notes:

The driver INF will also need to be modified with the following changes to support event notifications:

Use AddReg directive to reference a new add-registry-section to add endpoint property keys. This step can be skipped if such a section already exists. The following example adds a new add-registry-section (HDAudio.EPPProperties.AddReg) under HdAudModel.PrimarySpeakerTopo:

```
[HdAudModel.PrimarySpeakerTopo]
```

```
AddReg = HDAudio.EPPProperties.AddReg
```

Next, create an add-registry-section, if it does not already exist, to add endpoint property keys to the registry. Example below adds the appropriate property key for the first endpoint declared in this INF:

```
[HDAudio.EPPProperties.AddReg]
```

```
HKR,"EP\0",%PKEY_AudioEndpoint_Association%,%KSNODETYPE_ANY%
```

HKR,"EP\0",%PKEY_AudioEndpoint_Supports_EventDriven_Mode%,0x00010001,0x1

In the strings section, add the following section for the value of the property keys used in Step 2 above:

PKEY_AudioEndpoint_Association = "{1DA5D803-D492-4EDD-8C23-E0C0FFEE7F0E},2"

PKEY_AudioEndpoint_Supports_EventDriven_Mode = "{1DA5D803-D492-4EDD-8C23-E0C0FFEE7F0E},7"

Please reference Windows Driver Kit documentation for details on writing an INF file.

For more details, see "A Wave Port Driver for Real-Time Audio Streaming" at <http://go.microsoft.com/fwlink/?LinkId=40502>.

Device.Audio.HardwareAudioProcessing

HardwareAudioProcessing

Device.Audio.HardwareAudioProcessing.AudioHardwareOffloading

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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.

Device.Audio.HardwareAudioProcessing.Loopback

Validate Audio loopback attenuation and optional loopback tap point selection.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If hardware loopback is supported, the driver must always support loopback audio without volume or mute applied. If only this pre-volume/mute loopback type is supported, the loopback tap point capability is optional.

If the driver chooses to also provide loopback audio with mute and volume applied, the driver must implement the loopback tap point capability to communicate the ability. It must indicate that both pre-volume/mute and post-volume/mute loopback types are supported.

Device.Audio.HDAudio

This audio device uses the HD audio driver.

Device.Audio.HDAudio.HDAudioSpecCompliance

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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.

1. 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.
2. 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.
3. 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).
4. 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.
5. Codecs must respond to a verb even if addressed at a nonexistent widget or if the verb itself is invalid.
6. Function group nodes must have node IDs in the range 0 to 127. This restriction does not apply to node IDs for widget nodes.
7. 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).
8. 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.
9. 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.

10. 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:
 - a. Remembered or discovered the current power state of the function group
 - b. 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:

1. Intel High Definition Audio Controller specification
2. Be updated to comply with future specification revisions
3. Comply with the latest HD Audio specification ECRs in accordance with policies around new hardware requirements.
4. 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.
5. A system with HD Audio Controller hardware that declares support for a non-zero Programming Interface code in its Compatible ID list must also include a third-party controller driver for that hardware. An example Compatible ID with non-zero Programming Interface code would be PCI\CC_0403**10**. This requirement applies to new systems shipping with the Windows Oct 2021 Release and does not apply to systems that originally shipped with any previous Windows Release.

Device.Audio.USB

This audio device uses the USB audio driver.

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 11 Client x64
Windows 11 Client ARM64

Description

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 11 Client x64 Windows 11 Client ARM64

Description

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/

Device.BusController

In the next Windows major release (2022), the following requirements will become mandatory:

- **PLDR (Platform Level Device Reset)**

System : System.Client.BluetoothController.ErrorRecovery.CoreRequirement

PLDR is a very important feature that will help Bluetooth device recovery. **This will be a mandatory requirement starting in the next Windows major release in 2023.**

OEMs would be required to meet this requirement to qualify for respective WHCP certification.

Device.BusController.Bluetooth.Base

Device.BusController.Bluetooth.Base.4LeSpecification

Bluetooth controllers must support the Bluetooth 4.0 specification requirements. Support for Bluetooth 4.1 specification with LE Link Layer Topology is recommended.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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.

It is highly recommended to support Bluetooth 4.1 with LE Link Layer Topology which provides better proximal experiences and cross-device connectivity over Bluetooth LE.

"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.LeWhiteList

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description

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

Device.BusController.Bluetooth.Base.NoBluetoothLEFilterDriver

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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 all informational parameters commands as described in the Bluetooth Core version 5 specification, Part E, Section 7.4 and Erratum 10734.

In addition to the above, the following features shall be supported:

Bit position	Link Layer Feature
0	LE Encryption

Device.BusController.Bluetooth.Base.SupportsBluetooth21AndEdr

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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).

Device.BusController.Bluetooth.WidebandSpeech

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description:

Wideband speech enables high definition voice quality (audio is sampled at 16 KHz as opposed to only 8 KHz) for telephony audio on Windows devices when the user is communicating via a Bluetooth peripheral that also supports wideband speech.

What this means is that Bluetooth radios must satisfy wideband speech requirements as defined by the Bluetooth SIG [Hands-Free Profile \(HFP\) 1.6 specification](#).

Additionally for Windows Mobile:

- Radios must follow the [Core Specification Addendum \(CSA\) 2](#), which is included in the [Core Version 4.1](#) Bluetooth specification.
- At a minimum radios must support at least one Bluetooth SIG defined wideband speech codec (currently mSBC).

Business Justification:

We want users to experience the best possible quality audio when using Bluetooth peripherals on Windows. Wideband speech is becoming a standard for peripherals that support the HFP profile. Our competition already supports it

Device.BusController.Bluetooth.Base.Audio

This section contains requirements for Bluetooth audio.

Device.BusController.Bluetooth.Base.Audio.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

This requirement will be enforced as “If Implemented” from Feb 2022 onwards.

Bluetooth audio A2dp and HFP should play. Additionally, they should be able to be torn down and brought back up multiple times.

Device.BusController.Bluetooth.CoreSpecification4Dot1.LELinkLayerTopology

These requirements apply to Bluetooth controllers implementing LE Link Layer Topology as defined in the Bluetooth 4.1 specification.

Device.BusController.Bluetooth.CoreSpecification4Dot1.LELinkLayerTopology.CoreRequirement

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description

The Bluetooth controller must allow the spec-defined LE state combinations (as allowed in section [Vol 6] Part B, Section 1.1.1 of the Bluetooth Core version 5.0 specification): Only the following states are not required to be supported:

- Any combination of states that include the Low Duty Cycle Directed Advertising state.
- Any combination of states that include the High Duty Cycle Directed Advertising state

Device.BusController.Bluetooth.CoreSpecification5.LEExtendedAdvertising

These requirements apply to Bluetooth controllers that implement the LE Extended Advertising feature as defined in the Bluetooth 5.0 specification.

Device.BusController.Bluetooth.CoreSpecification5.LEExtendedAdvertising.CoreRequirement

Bluetooth controllers must support the LE Extended Advertising feature as defined in the Bluetooth 5.0 specification.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

Windows will not use the LE Extended Advertising feature unless support is declared and this requirement is satisfied. To declare support, include the following in the DDInstall.HW section of the INF file for the Bluetooth controller:

```
Include = bth.inf
Needs = BthLEExtendedAdvertisingOptIn.HW
```

The controller must also implement support to enable the Link Layer Features as defined in the Bluetooth 5.0 specification:

Bit Position	Link Layer Feature
12	LE Extended Advertising

Device.BusController.Bluetooth.CoreSpecification5Dot2.AutoConnectionEstablishmentProcedure

These requirements apply to Bluetooth controllers implementing LE Auto Connection Establishment Procedure as defined in the Bluetooth 5.2 specification.

Device.BusController.Bluetooth.CoreSpecification5Dot2.AutoConnectionEstablishmentProcedure.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

This requirement will be enforced as "If Implemented" from Feb 2022 onwards.
Windows will not use the LE Auto Connection Establishment Procedure feature unless support is declared and this requirement is satisfied. To declare support, include the following in the DDInstall.HW section of the INF file for the Bluetooth controller:

```
Include = bth.inf  
Needs = BthLEAutoConnectionEstablishmentOptIn.HW
```

Device.BusController.Bluetooth.CoreSpecification5Dot2.LEPowerControl

This section contains requirements for LE Power Control feature as defined in the Bluetooth Core Specification 5.2 and any erratas thereafter.

Device.BusController.Bluetooth.CoreSpecification5Dot2.LEPowerControl.CoreRequirement

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented

Description

Implement LE Power Control feature that was added in the Bluetooth Core Specification version 5.2 and any erratas since then.

This requirement refers to bit 33 and 34 of the Link Layer supported features as defined in BLUETOOTH SPECIFICATION Version 5.2 | Vol 6, Part B, section 4.6 FEATURE SUPPORT.

Device.BusController.Bluetooth.CoreSpecification5Dot3.LEEnhancedConnectionUpdate

This section contains requirements for LE Enhanced Connection Update feature as defined in the Bluetooth Core Specification 5.3 and any erratas thereafter.

Device.BusController.Bluetooth.CoreSpecification5Dot3.LEEnhancedConnectionUpdate.CoreRequirement

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented

Description

Implement LE Enhanced Connection Update feature that was added in the Bluetooth Core Specification version 5.3 and any erratas since then.

This requirement refers to bit 37 and 38 of the Link Layer supported features as defined in BLUETOOTH SPECIFICATION Version 5.3 | Vol 6, Part B, section 4.6 FEATURE SUPPORT.

Device.BusController.Bluetooth.CoreSpecification5Dot3.LEChannelClassification

This section contains requirements for LE Channel Classification feature as defined in the Bluetooth Core Specification 5.3 and any erratas thereafter.

Device.BusController.Bluetooth.CoreSpecification5Dot3.LEChannelClassification.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

Implement LE Channel Classification feature that was added in the Bluetooth Core Specification version 5.3 and any erratas since then.

This requirement refers to bit 39 of the Link Layer supported features as defined in BLUETOOTH SPECIFICATION Version 5.3 | Vol 6, Part B, section 4.6 FEATURE SUPPORT.

Device.BusController.Bluetooth.ErrorRecovery

These requirements apply to internal Bluetooth controllers that support one or more platform-defined error recovery mechanisms.

Device.BusController.Bluetooth.ErrorRecovery.CoreRequirement

Internal Bluetooth controllers and the platforms they are present on must support one or more platform-defined error recovery mechanisms to be able to recover from certain error conditions.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

Platforms that support internal Bluetooth controllers must support one or more platform-defined error recovery mechanisms as documented in [MSDN](#). Specifically, support for Platform-Level Device Reset (PLDR) and/or Function-Level Device Reset (FLDR) is required. Supporting these mechanisms allows the Bluetooth stack to recover the stack from certain otherwise irrecoverable error conditions by attempting to reset the Bluetooth controller in order to return to a clean working state.

Device.BusController.Bluetooth.HCIExtensions1

These requirements apply to Bluetooth controllers that implement the Microsoft Bluetooth HCI Extensions version 1.

Device.BusController.Bluetooth.HCIExtensions1.CoreRequirement

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

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Description

Radios must comply with the Microsoft defined Bluetooth HCI extensions specification. The details of the specifications are located on MSDN: <https://aka.ms/bthciextensions>

The following supported features are required:

Value	Parameter description
0x00000000 00000001	Controller supports the RSSI Monitoring feature for BR/EDR connections. In addition, the controller supports HCI VS MSFT Read Absolute RSSI to read the absolute RSSI metric of a BR/EDR connection.
0x00000000 00000002	Controller supports the RSSI Monitoring feature for LE connections.
0x00000000 00000004	Controller supports the RSSI Monitoring of LE advertisements.
0x00000000 00000008	Controller supports Advertising Monitoring of LE advertisements.
0x00000000 00000020	Controller supports Continuous Advertising Monitoring of LE advertisements performed concurrently with other radio activities.

The following feature is required if the HCI commands added in Erratum 10734 are not supported:

0x00000000 00000010	Controller supports verifying the validity of the public X and Y coordinates on the curve during the Secure Simple pairing process for P-192 and P-256. For more information, see Bluetooth Core Specification Erratum 10734 .

Device.BusController.Bluetooth.HciExtensions.A2dpOffload

This section contains requirements for Bluetooth A2DP offloading.

Device.BusController.Bluetooth.HciExtensions.A2dpOffload.Initialization

AVDTP offload commands should be supported if the radio supports A2DP offload.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

The IHV driver, Controller, and Windows stack should work together to bring up and tear down an A2DP offloaded stream. As subparts, if HCI_VS_MSFT_Read_Supported_Features has the offload command bit set, then HCI_VS_MSFT_Avdtp* HCI commands should exist and if the IHV audio driver and the controller both support A2DP offloading then the host should express support.

The tests for this requirement may need special hardware to execute. Please see <https://aka.ms/bthlksetup> for more information.

Device.BusController.Bluetooth.HciExtensions.A2dpOffload.Streaming

AVDTP offload commands should be supported if the radio supports A2DP offload.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

This requirement will be enforced as “If Implemented” from Feb 2022 onwards.
The A2DP audio stream should be able to stream audio data with few to no glitches.

The tests for this requirement may need special hardware to execute. Please see <https://aka.ms/bthlksetup> for more information.

Device.BusController.Bluetooth.HciExtensions.A2dpOffload.ReportedCodecCapabilities

Bluetooth controller should only report known codecs.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

This requirement will be enforced as “If Implemented” from Feb 2022 onwards.

Bluetooth controller and IHV audio driver MUST report SBC as one of the supported codecs. Additional codecs reported MUST be from the list below (list can be appended in new releases):

- AAC
- aptX

The tests for this requirement may need special hardware to execute. Please see <https://aka.ms/bthlksetup> for more information.

Device.BusController.Bluetooth.HciExtensions.A2dpOffload.SimultaneousSco

IHV Audio driver and Bluetooth Controller should ensure that A2DP offload should suspend if SCO is active.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

This requirement will be enforced as “If Implemented” from Feb 2022 onwards.

IHV audio driver and Bluetooth Controller should stop streaming A2DP audio data when told to suspend and should restart when told to restart. This is a common scenario when a SCO channel is opened/closed e.g. when a user answers a phone call while streaming music.

The tests for this requirement may need special hardware to execute. Please see <https://aka.ms/bthlksetup> for more information.

Device.BusController.Bluetooth.HciExtensions.MonitorAdvertisementV2

This section contains requirements for Microsoft .Vendor HCI Extensions to filter extended advertisements.

Device.BusController.Bluetooth.HciExtensions.MonitorAdvertisementV2.CoreRequirements

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

Implement Microsoft vendor HCI extensions to filter extended advertisement filter.

Device.BusController.Bluetooth.LEAudioBroadcast

This section contains requirements for LE Audio Broadcast feature. This is an optional feature, but all requirements under this section must be implemented if LE Audio Broadcast feature is supported.

To declare support, include the following in the DDInstall.HW section of the INF file for the Bluetooth controller:

```
Include = bth.inf  
Needs = BthLEBroadcastIsochronousStreamsOptIn.HW
```

Also, the following must be supported:

- WHCP Device.BusController.Bluetooth.LEAudioUnicast feature
- Link Layer features as defined in BLUETOOTH SPECIFICATION Version 5.4 | Vol 6, Part B, section 4.6 FEATURE SUPPORT:

Bit Position	Link Layer Feature
13	LE Periodic Advertising
24	Periodic Advertising Sync Transfer - Sender
25	Periodic Advertising Sync Transfer - Recipient
30	Isochronous Broadcaster
31	Synchronized Receiver

Device.BusController.Bluetooth.LEAudioBroadcast.LEAudioUnicast

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Description

Required if the LE Audio Broadcast feature is to be supported.

WHCP feature Device.BusController.Bluetooth.LEAudioUnicast shall be supported and all associated requirements satisfied.

Device.BusController.Bluetooth.LEAudioBroadcast.PeriodicAdvertising

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Description

Required if the LE Audio Broadcast feature is to be supported.

As defined in BLUETOOTH SPECIFICATION Version 5.4 | Vol 6, Part B, section 4.6 FEATURE SUPPORT, the following Link Layer features shall be supported::

Bit Position	Link Layer Feature
13	LE Periodic Advertising
24	Periodic Advertising Sync Transfer - Sender
25	Periodic Advertising Sync Transfer - Recipient

Device.BusController.Bluetooth.LEAudioBroadcast.BroadcastIsochronousStreams

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Description

Required if the LE Audio Broadcast feature is to be supported.

As defined in BLUETOOTH SPECIFICATION Version 5.4 | Vol 6, Part B, section 4.6 FEATURE SUPPORT, the following Link Layer features shall be supported:

Bit Position	Link Layer Feature
30	Isochronous Broadcaster
31	Synchronized Receiver

Device.BusController.Bluetooth.LEAudioUnicast

This section contains requirements for LE Audio Unicast feature. This is an optional feature, but all requirements under this section must be implemented if LE Audio Unicast feature is supported.

To declare support, include the following in the DDInstall.HW section of the INF file for the Bluetooth controller:

```
Include = bth.inf
Needs = BthLEConnectedIsochronousStreamsOptIn.HW
```

Device.BusController.Bluetooth.LEAudioUnicast.HciExtensionsMonitorAdvertisementV2

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

WHCP feature Device.BusController.Bluetooth.HciExtensions.MonitorAdvertisementV2 must be supported.

Device.BusController.Bluetooth.LEAudioUnicast.BRSecureConnections

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: If-Implemented

Description

Implement BR secure connections feature that was added in the Bluetooth Core Specification version 4.1 and any erratas since then.

This requirement refers to feature number 51, 64, 67, and 136 in BLUETOOTH SPECIFICATION Version 4.1 | Vol 2, Part C, section 3.3 FEATURE MASK.

Required if the LE Audio Unicast feature is to be supported.

Device.BusController.Bluetooth.LEAudioUnicast.LEConnectedIsochronousStreams

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description

This requirement refers to bit 28, 29, and 32 of the Link Layer supported features as defined in BLUETOOTH SPECIFICATION Version 5.2 | Vol 6, Part B, section 4.6 FEATURE SUPPORT.

Required if the LE Audio Unicast feature is to be supported.

[**Device.BusController.Bluetooth.LEAudioUnicast.LEPowerControl**](#)

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented

Description

WHCP feature Device.BusController.Bluetooth.CoreSpecification5Dot2.LEPowerControl must be supported.

Required if the LE Audio Unicast feature is to be supported.

[**Device.BusController.Bluetooth.LEAudioUnicast.LEEnhancedConnectionUpdate**](#)

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented

Description

WHCP feature Device.BusController.Bluetooth.CoreSpecification5Dot3.LEEnhancedConnectionUpdate must be supported.

Required if the LE Audio Unicast feature is to be supported.

[**Device.BusController.Bluetooth.LEAudioUnicast.LEChannelClassification**](#)

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented

Description

WHCP feature Device.BusController.Bluetooth.CoreSpecification5Dot3.LEChannelClassification must be supported.

Required if the LE Audio Unicast feature is to be supported.

Device.BusController.Bluetooth.LEAudioUnicast.CoreSpecification5Dot3

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented**Description**

Controller shall be Bluetooth Core Specification.version 5.3 compliant and shall reflect this in the returned parameters of the HCI_Read_Local_Version_Information command's HCI_Version and LMP_Version.

Device.BusController.Bluetooth.LEAudioUnicast.Transport

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented**Description**

Implement BR secure connections feature that was added in the Bluetooth Core Specification version 4.1 and any erratas since then.

This requirement refers to feature number 51, 64, 67, and 136 in BLUETOOTH SPECIFICATION Version 4.1 | Vol 2, Part C, section 3.3 FEATURE MASK.

Required if the LE Audio Unicast feature is to be supported.

Device.BusController.Bluetooth.MicrosoftStack

Device.BusController.Bluetooth.MicrosoftStack.BluetoothDeviceClass

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Description

Formerly Device.BusController.Bluetooth.Base.MicrosoftBluetoothStack.

The Microsoft Bluetooth stack must be present if there are any devices with the Bluetooth device class.

Device.BusController.Bluetooth.NonUSB

Bluetooth Controller - NonUSB connected radios

Device.BusController.Bluetooth.NonUSB.Performance

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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).

Device.BusController.Bluetooth.USB

Bluetooth Controller - USB connected radios

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 11 Client x64 Windows 11 Client ARM64 Windows Server x64

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.

Device.BusController.Bluetooth.HciExtensions.EnhancedControllerDiagnostics

These requirements apply to Bluetooth controllers implementing Enhanced Controller Diagnostics portion of Microsoft defined Hci Extensions.

Device.BusController.Bluetooth.HciExtensions.EnhancedControllerDiagnostics.CoreRequirement

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: If-Implemented

Description

This requirement will be enforced as “If Implemented” from Feb 2022 onwards.
This requirement enforces sequence of vendor specific HCI commands/events needed to enable the Bluetooth controller to provide diagnostic information to Windows.

The following supported features are required:

Value	Parameter description
0x00000000 00000040	Controller diagnostic TLV
0x00000000 00000200	Enhanced controller diagnostic TLV

Device.BusController.Bluetooth.UsbTransport.Transport

This section contains requirements for any Bluetooth controller which communicates with the host computer via the Host Controller Interface (HCI) over the Universal Serial Bus (USB).

Device.BusController.Bluetooth.UsbTransport.Transport.UsbDescriptors

Applies to

Windows 11 Client x64
Windows 11 Client ARM64
Windows Server x64

Terms: Optional

Enhancement Scenarios: Ensure a USB-based Bluetooth controller can be discovered and enumerated correctly.

Description

A controller shall report the attributes of its USB interface via USB descriptors. For more information, see section 9.6 “Standard USB Descriptor Definitions” of [USB 2.0 Specification](#) or [USB 3.2 Specification](#).

The USB interface shall support necessary endpoints which meet the requirements as described in section 2 “USB ENDPOINT EXPECTATIONS” of [Bluetooth Core Specification version 5.3 | Vol 4 \(Host Controller Interface\), Part B \(USB Transport Layer\)](#). In addition, the maximum packet size for an isochronous endpoint shall have the same value as its “suggested max packet size” counterpart. For example, the maximum packet sizes for alternate setting 2 and 6, if

implemented, must be 17 and 63 respectively (see Table 2.1 “USB firmware interface and endpoint settings” in Bluetooth Core Specification | Vol 4, Part B).

Device.BusController.Bluetooth.UsbTransport.AlternateSetting6

This section contains requirements for those USB-based Bluetooth controllers which support alternate setting 6.

Device.BusController.Bluetooth.UsbTransport.AlternateSettings6.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Terms: Optional

Enhancement Scenarios: Enable framed data such as mSBC wideband speech audio frames or other audio codec frames over USB alternate setting 6 for the *hands-free profile* (HFP).

Description

Alternate setting 6 shall be implemented in such a way that the controller guarantees the following:

- Given an outgoing HCI packet sent by the host, the first octet in its payload shall be transmitted as the first octet in the payload of an *enhanced synchronous connection-oriented* (eSCO) packet.
- Each incoming USB isochronous packet received by the host, if present, shall contain a full maximum packet size's worth of data. (In other words, each packet should be 63 bytes in length for USB alternate setting 6 assuming full-speed USB). If the actual packet size is different from maximum packet size, the host may drop the data.

Device.BusController.I2C

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

Device.BusController.I2C.CancellationOfIO

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

Applies to	Windows 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 (100 Kbps), fast (400 kbps) 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 11 Client x64

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 11 Client x64

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 (100 Kbps), fast (400 kbps) 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 11 Client x64

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.

Device.BusController.I3C

These requirements apply only to I3C controller silicon vendors. System manufacturers may optionally run these tests, but may need hardware customization.

Device.BusController.I3C.DirectMemoryAddressingIO

I3C controller must support direct memory addressing (DMA) I/O

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

I3C controller must support DMA I/O.

Device.BusController.I3C.HLKTestability

Systems with I3C controllers must expose correct ACPI table information and I3C pin-outs to enable HLK testability.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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

Details:

- Controller under test must provide I3C external connectivity pin-out (SCL, SDA, Power and GND).
- When testing an I2C peripheral on the I3C controller, update ACPI to correctly describe I2C HLK test peripheral drivers, its connection to I3C controller under test, and the fact that I2C devices are present.
- When testing a static-only I3C peripheral on the I3C controller, update ACPI to correctly describe required I3C device properties (static address).
- Other peripheral devices on the same I3C controller under test must be disabled when running HLK tests.

Device.BusController.I3C.IdlePowerManagement

I3C controller ~~and controller drivers~~ must support Idle Power Management.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support the following:

- Controller should go to the DX state after it is idle for more than 1 second when the screen is on.
- Controller should go to the DX 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 DX state to the D0 state.

Note: Idle timeouts are subject to change as Microsoft evaluates power and battery life requirements

Device.BusController.I3C.MIPIConformanceTests

The I3C controller must be tested against the MIPI I3C conformance tests. The controller must be compliant to version MIPI I3C HCI specification version 1.0 or higher.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must pass the latest [MIPI I3C Conformance Tests Suite for controllers](#).

The controller must be compliant to version MIPI I3C HCI specification version 1.0 or higher.

Device.BusController.I3C.RemoteWake

I3C controller must support the ability for a peripheral to remotely wake the controller and system

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support remote wake from peripherals to wake the controller and the system. For example, if an I3C touchpad is connected to the system and the user clicks the touchpad while the system is asleep (triggering an IBI), the system must wake up.

Device.BusController.I3C.SleepTests

The I3C controller must be capable of powering down when it is idle and resuming from a low power state.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must be capable of powering down when it is idle and resuming from a low power state. The host controller must enter its lowest possible power state when idle. On resume from sleep the host controller and its devices on the bus should resume operation transparently to the end-user.

Device.BusController.I3C.SystemPowerManagement

I3C controller must support all system power states.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support all system power states. For example, if an I3C peripheral is connected to the system before the system goes to sleep, the I3C peripheral must continue to work after the system comes out from sleep.

Device.BusController.I3C.TestedUsingMicrosoftI3CDrivers

The use of the Microsoft provided HCI driver is required.

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

The use of the Microsoft provided HCI driver is required.

Device.BusController.I3C.I2C

Device.BusController.I3C.I2C.CancellationOfIO

I3C controller must support the cancellation of I/O requests.

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

If-implemented I2C support on an I3C controller, the I3C controller must conform to the SPB framework and support the following requirements for I2C peripherals:

- Driver implements SPB request cancellation logic for read/write/sequence I/O.
- Cancellations complete in < 1 second

Device.BusController.I3C.I2C.LockUnlockIOCTL

I3C controller must support the I2C Lock/Unlock IOCTL.

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

If-implemented I2C support on an I3C controller, the I3C controller must conform to the SPB framework and support the following for I2C peripherals:

- 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.I3C.I2C.NACK

I3C controller must support I2C peripheral NACK.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If-implemented I2C support on an I3C controller, the I3C controller must conform to the SPB framework and support the following for I2C peripherals:

- 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.I3C.I2C.SPBCx

I2C peripherals connected to the I3C controller must use the SPBCx framework.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If-implemented I2C support on an I3C controller, I2C peripherals connected to the I3C controller must use the SPBCx framework. All I2C peripheral device drivers on an I3C controller bus will hook into the Microsoft provided I3C HCI driver.

Device.BusController.I3C.I2C.SPBRead

I3C controller must support I2C SPB Read operations correctly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If-implemented I2C support on an I3C controller, the I3C controller must conform to the SPB framework and support the following when reading data for I2C peripherals:

- Must support reading from Fm (400 kbps) and Fm+ (1 Mbps) peripheral targets, with appropriate timing for addressing/data.
- Must support read size from 1 to 4096 bytes (4 KBytes).
- Sizes larger than 4 KBytes must succeed or fail with STATUS_NOT_SUPPORTED.
- Fail any unsupported data size read request with STATUS_INVALID_PARAMETER and not cause any bus activities.

Device.BusController.I3C.I2C.SPBSequenceIOCTL

I3C controller and controller drivers must support I2C SPB Sequence IOCTL correctly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If implemented I2C support on an I3C controller, the I3C controller must conform to the SPB framework and support the following I2C peripherals:

- Supports any arbitrary I/O sequences: write-read, read-write, write-write, read-read and complex combined such as write-read-read-write-write
- 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.I3C.I2C.SPBWrite

I3C controller must support I2C SPB Write operations correctly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If implemented I2C support on an I3C controller, the I3C controller must conform to the SPB framework and support the following when writing for I2C peripherals:

- Must support writing to Fm (400 kbps) and Fm+ (1 Mbps) peripherals.
- Must support write size from 1 to 4096 bytes (4 KBytes).
- Sizes larger than 4 KBytes must succeed or fail with STATUS_NOT_SUPPORTED.
- Fail any unsupported data size write request with STATUS_INVALID_PARAMETER and not cause any bus activities.

Device.BusController.I3C.I2C.Stress

I3C controller must operate correctly and recover from bus hangs or faults under prolonged stress conditions with I2C peripherals.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If implemented I2C support on an I3C controller, the I3C controller must conform to the SPB framework and support the following for I2C peripherals:

- Supports bus recovery when peripheral(s) hang.

- Sustain multiple targets stress for more than 1 hour.

Device.BusController.I3C.I2C.ValidI2C

I2C peripherals connected to the I3C controller must follow constraints set by the I3C specification.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If implemented I2C support on an I3C controller, I2C peripherals connected to the I3C controller must follow constraints set by the I3C specification. These include the following:

- Must have a static I2C address.
- Must support at least Fm speed.
- Cannot perform clock stretching.
- Cannot use reserved I3C addresses.

Device.BusController.I3C.I3C

Device.BusController.I3C.I3C.CancellationOfIO

I3C controller must support the cancellation of I/O requests.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support the following requirements:

- Driver implements request cancellation logic for read/write/sequence I/O.
- Cancellations complete in < 1 second

Device.BusController.I3C.I3C.DynamicAddressing

I3C controller must support dynamic addressing to enable dynamic enumeration.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support dynamic addressing which enables discovering and reporting connected I3C targets to the I3C controller driver at runtime.

Device.BusController.I3C.I3C.InBandInterrupts

I3C controller must support in-band interrupts.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

I3C controller must support in-band interrupts to allow target(s) to signal when new data is available using the IBI method in I3C.

Device.BusController.I3C.I3C.MaxReadWriteLength

I3C controller must support getting/setting max read/write lengths correctly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support setting and getting the max read and write lengths (SETMWL, GETMWL, SETMRL, GETMRL).

Device.BusController.I3C.I3C.NACK

I3C controller must support target NACK.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must 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 read 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 read.
- Controller can detect device NACK during a read operation of a sequence, complete the request with STATUS_SUCCESS, and information bytes is set to number of bytes that is less than what was intended to be read.

Device.BusController.I3C.I3C.NoActiveControllerHandoff

I3C controller must be the Primary and Active Controller and not advertise capability of handoff of Active Controller status.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

I3C controller must always be the Primary and Active Controller on the bus per I3C specification definition.

I3C controller must not advertise handoff of Active Controller support to other possible peripherals on the bus. No other devices connected to the bus should advertise handoff of Active Controller.

Device.BusController.I3C.I3C.Read

I3C controller must support Read operations correctly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support the following when reading data from an I3C target:

- Must support reading at least 1 MHz SDR from targets.
- Must support I3C Open Drain (Addressing) SDR and Push Pull (Data) SDR transfer modes. Recommended to also support the HDR transfer modes.
- Must support read size from 1 to 4096 bytes (4 KBytes).
- Sizes larger than 4 KBytes must succeed or fail with STATUS_NOT_SUPPORTED.
- Fail any unsupported data size read request with STATUS_INVALID_PARAMETER and not cause any bus activities.

Device.BusController.I3C.I3C.Sequence

I3C controller must support an I/O Sequence correctly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must 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
- Controller needs to examine the sequence and determine if it is supported or fail with STATUS_INVALID_PARAMETER before causing any bus activities.

Device.BusController.I3C.I3C.StaticAddressing

I3C controller must support static addressing.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The I3C controller must support static addressing (i.e. describing connected targets in ACPI).

Device.BusController.I3C.I3C.Stress

I3C controller must operate correctly and recovers from bus hangs or faults under prolonged stress conditions.

Applies to	Windows 11 Client x64 <u>Windows 11 Client ARM64</u>

Description

The I3C controller must support the following:

- Supports bus recovery when target(s) hang.
- Sustain multiple targets stress for more than 1 hour.

Device.BusController.I3C.I3C.TargetReset

I3C controller must support resetting an individual target.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

I3C controller must support resetting an individual target on the bus.

Device.BusController.I3C.I3C.Write

I3C controller must support Write operations correctly.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Established: 2302

Description

The I3C controller must support the following when writing to an I3C target:

- Must support writing at least 1 MHz SDR to a target.
- Must support I3C Open Drain SDR and Push Pull SDR transfer modes. Recommended to also support the HDR transfer modes.
- Must support write size from 1 to 4096 bytes (4 KBytes).
- Sizes larger than 4 KBytes must succeed or fail with STATUS_NOT_SUPPORTED.
- Fail any unsupported data size write request with STATUS_INVALID_PARAMETER and not cause any bus activities.

Device.BusController.NFC

It is strongly recommended that IHV's create NFC client drivers that interface with the [Microsoft NFC Class Extension \(CX\)](#).

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](#).

Device.BusController.NFC.NearFieldProximity.Base

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.NDEF

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The NFP provider must be able to receive, handle and transmit native NDEF packets via Logical Link Control Protocol (LLCP) v1.1 and Simple NDEF Exchange Protocol (SNEP) v1.0 defined by NFC Forum.

Device.BusController.NFC.NearFieldProximity.PublishSubscribe

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The NFP provider must be able to receive and deliver messages using subscription and publication.

Device.BusController.NFC.NearFieldProximity.TagOperation

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The NFP provider must be able to reliably read, write, and format tags as well as set tags to read only.

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](#).

Device.BusController.NFC.RadioManagement.Base

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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

Device.BusController.NFC.CardEmulation

Terms: If-Implemented

The Secure Element DDI allows callers to the NFC device driver to enumerate, communicate with, and configure the secure elements accessible from the device, including Host Card Emulation (HCE) based secure elements when data is routed directly to applications running on the host CPU.

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](#).

Device.BusController.NFC.CardEmulation.Base

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

An NFC Secure Element provider that supports card emulation must be able to enumerate all secure elements that are accessible from the device, report NFCC capabilities and enforce exclusive access to the client to manage card emulation mode.

Device.BusController.NFC.CardEmulation.Event

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

An NFC Secure Element provider that supports 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 and HCE being activated or deactivated.

Device.BusController.NFC.CardEmulation.Communication

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

An NFC Secure Element provider that supports card emulation must be able to reliably route data between contactless interface and the host or other secure element appropriately based on the current listen mode routing table configuration.

Device.BusController.NFC.CardEmulation.RoutingTable

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

An NFC Secure Element provider that supports card emulation must be able to configure the listen mode routing table for all secure elements accessible from the device.

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](#).

Device.BusController.NFC.SmartCard.ResourceManagement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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

Device.BusController.NFC.SmartCard.Communication

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The Smart Card Reader provider must be able to communicate according to PC/SC standards with supported contactless smart cards. It must include support for PC/SC transparent mode. Supported contactless smart cards include: MIFARE Ultralight family, MIFARE Desfire family, Topaz, Jewel, ISO14443-A/B, Felica, ISO15693, MIFARE Classic (optional).

Device.BusController.SdioController

Device.BusController.SdioController.ComplyWithIndustrySpec

SDIO controller must comply with the industry standard.

Applies to	Windows 11 Client 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 11 Client x64

Description

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

Device.BusController.UART

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

Device.BusController.UART.Cancellation

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

Applies to	Windows 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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.GuardBuffer

UART controller and associated controller driver must detect and correct the guard buffer's overflow and underflow.

Applies to	Windows 11 Client x64

Description

The UART controller and associated controller driver must conform to the Serial framework and should not write to memory outside of the transfer buffer. For example, if a 1KB data transfer buffer is allocated for a read, the UART data transfer must not write to memory outside of the 1KB buffer. For this requirement, a test has been added to detect if memory outside of the transfer buffer has been overwritten. Once the transfer has completed, ownership of the transfer buffer is handed to the test application. The test then verifies the transfer buffer memory is not altered to ensure the buffer is not being overwritten.

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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.

Device.BusController.UsbController

Device.BusController.UsbController.ImplementAtLeastOneXhciSpcStructForUSB2

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

USB host controllers must pass USB Implementers Forum (IF) testing. For details, see the following link:

- [https://msdn.microsoft.com/en-us/library/windows/hardware/dn434058\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/dn434058(v=vs.85).aspx)

USB Type-C PD silicon (e.g. PD controller) must implement version 1.1 of the USB Type-C spec or higher and revision 2.0 v1.1 and optionally support revision 3.0 v1.0 or higher of the PD spec and must be certified according to the USB-IF's USB-C product testing matrix.

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 WLK 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 5.5.2.3.2).

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 11 Client 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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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. In 2019 Microsoft will transition to the text of this requirement being "USB 3.1 controllers must comply with the Extensible Host Controller Interface (xHCI) Specification version 1.1 and any USB-IF Errata that are released by the USB-IF. ". Microsoft strongly encourages transitioning to being compliant with the specification version 1.1 before the 2019 date.

Device.BusController.UsbController.USB4

This section contains requirements for USB4 Host Routers.

Device.BusController.UsbController.USB4.DomainSleep

USB4 Host Router supports Sleep. USB4 host router(s) must power down when all attached device(s) have been idle for some time.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

USB4 Host Routers must reliably support USB4 Sleep and entry and exit when all tunneled protocols are idle. Tunneled device protocols must not electrically signal disconnect to their respective tunnels on entry or exit from USB4 Sleep. And the USB4 Host Router must replay the protocol wake over the appropriate tunnel after the USB4 domain has resumed from Sleep.

Device.BusController.UsbController.USB4.HostRouterSupportsECNsFromUsbif

USB4 Host Router supports USB4 ECNs from the USB-IF

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

USB4 Host Routers must support the USB4 ECNs as defined by the USB-IF in the base USB4 specification at TBD version/date.

Device.BusController.UsbController.USB4.HostRouterSupportsPCleTunneling

USB4 Host Router supports tunneling PCIe protocol.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

USB4 Host Routers must support tunneling PCI Express (PCIe) in accordance with chapter 11 of the USB4 Specification and in compliance with the PCI Express Specification.

Extensibility of PCIe via the USB4 connector enables scenarios such as external graphics processing units (GPUs) and high-performance storage and is a key motivator of the USB4 technology. Users will expect their USB4 docks and peripherals with PCIe functions to work on any USB4 host. Thus, with the goal of maximizing compatibility across the ecosystem, all USB4 Host Routers must support tunneling the PCIe protocol.

Device.BusController.UsbController.USB4.HostRouterSupportsThunderbolt3Compatibility

USB4 Host Router supports backwards compatibility with Thunderbolt® 3.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

USB4 Host Routers must support interoperability with Thunderbolt® 3 (TBT3) peripherals according to chapter 13 of the USB4 specification. Specifically, the USB4 Host Router must be interoperable with TBT3 Device Routers and cables; it is not required to be compatible with a TBT3 Connection Manager.

USB4 is an evolution of the TBT3 technology, using the same connector and filling many of the same high-performance requirements for users. As such, they will expect their existing TBT3 compatible peripherals to continue to function on their new USB4 enabled system. Thus, with the goal of maximizing compatibility with the existing TBT3 cable and peripheral ecosystem, all USB4 hosts must support interoperability with TBT3 Device Routers and cables.

Device.BusController.UsbController.USB4.TestedUsingMicrosoftUsbStack

USB4 Host and Device Routers are certified using the Microsoft provided USB4 drivers.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

USB4 Host and Device Routers must be tested with Microsoft's USB4 driver stack installed and enabled on a Windows system. All required features of the router must be functional and tested with the Microsoft USB4 driver.

Device.BusController.UsbController.USB4.DiscreteControllersSupportFPB

Discrete USB4 Controllers support FPB

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Hosts with discrete USB4 controllers must support a Flattening Portal Bridge (FPB) so that non-contiguous bus numbers and memory resources may allocated efficiently to child PCI devices. FPB is documented PCI Express Base Spec Rev 5.0 v1.0.

Device.BusController.UsbController.USBC

Device.BusController.UsbController.USBC.UsbTypeCAltModeCertification

USB Type-C Alternate Mode sub-systems are certified with their respective organizations.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If the USB Type-C PD silicon (e.g. PD controller) supports an Alternate Mode Standard and the industry group that owns that Standard has a corresponding certification, the silicon must get that certification, in addition to USB-IF certification.

For example, if USB Type-C PD silicon supports DisplayPort Alternate Mode in addition to USB and PD, then that silicon must be certified by VESA for the DisplayPort functionality, in addition to the USB-IF for USB and PD functionality.

Certification for vendor-specific Alternate Modes is not required.

Device.BusController.UsbController.USBC.UsbTypeCUCM

USB Type-C silicon supports UCM

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

This requirement applies to controllers that support Power Delivery or Dual Role or Alternate Modes.

USB Type-C PD silicon (e.g. PD controller) that run on a Windows 10 system and do not implement UCSI or UcmTcpci must ship with a 3rd-party UcmCx client Driver.

3rd-party UcmCx client drivers must be implemented as per the following MSDN documentation:

- [https://msdn.microsoft.com/en-us/library/windows/hardware/mt188011\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/mt188011(v=vs.85).aspx).

The UcmCx client driver is required to implement the following APIs:

UcmInitializeDevice

UcmConnectorCreate

UcmConnectorTypeCAttach

UcmConnectorTypeCDetach

UcmConnectorTypeCCurrentAdChanged

UcmConnectorChargingStateChanged

If the system or controller supports Power Delivery, the following additional APIs must be implemented:

UcmConnectorPowerDirectionChanged

UcmConnectorPdSourceCaps

UcmConnectorPdPartnerSourceCaps

UcmConnectorPdConnectionStateChanged

If the system or controller exposes dual role ports, the following additional APIs must be implemented:

UcmConnectorDataDirectionChanged

Device.BusController.UsbController.USBC.USBTypeCUCMTCPPI

USB Type-C silicon that support UCMTCPPI must implement it correctly

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

This requirement applies to controllers that support Power Delivery or Dual Role or Alternate Modes.

USB Type-C Port Controllers that are not paired with Type-C Port Manager silicon that provides the Type-C policy engine and protocol layer should implement a UcmTpcix client driver.

If the Type-C Port Controller uses UcmTpcpi, the UcmTpcix client driver is required to implement the following APIs:

- UcmTpcpiPortControllerCreate
- UcmTpcpiPortControllerSetHardwareRequestQueue
- UcmTpcpiPortControllerStart
- UcmTpcpiPortControllerAlert

Device.BusController.UsbController.USBC.UsbTypeCUCSI

USB Type-C silicon that support UCSI must implement UCSI correctly

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

This requirement applies to controllers that support Power Delivery or Dual Role or Alternate Modes.

USB Type-C PD silicon (e.g. PD controller) that is used with an Embedded Controller that implements the PD policy should implement UCSI.

If the USB Type-C PD silicon implements UCSI, it must implement UCSI v1.0 (or later). It must specify the UCSI version number that is implemented in the VERSION field of the UCSI data structure. In addition, the USB Type-C PD silicon must implement the following optional features from the UCSI spec.

Commands

- GET_ALTERNATE_MODES
- GET_CAM_SUPPORTED
- GET_PDOS
- GET_CABLE_PROPERTY
- SET_NOTIFICATION_ENABLE
 - The system or controller must support the following notifications within this command:
 - Supported Provider Capabilities Change
 - Negotiated Power Level Change

- Supported CAM Change
- GET_CONNECTOR_STATUS
 - The system or controller must support the following Connector Status Changes within this command:
 - Supported Provider Capabilities Change
 - Negotiated Power Level Change
 - The system or controller must support the following field in GET_CONNECTOR_STATUS Status structure
 - Provider Capabilities Limited Reason
 - Request Data Object

Device.Connectivity.BluetoothDevices

Devices that connect to the PC via Bluetooth.

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

The HIDReconnectInitiate attribute (defined in Bluetooth HID Profile, 1.0, Section 7.11.5, "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 11 Client x64 Windows 11 Client ARM64

Description

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

Device.Connectivity.BluetoothDevices.SupportBluetooth21

Devices that support Bluetooth must implement the Bluetooth 2.1 requirements.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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

Device.Connectivity.Network.VerticalPairing

Root for former Rally technologies

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.

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.msp>.

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 it 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.

Device.Connectivity.UsbDevices

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

Device.Connectivity.UsbDevices.DebugCompliesWithDebugSpec

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

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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.UsbifCertification

USB devices must either pass USB IF tests or be USB IF certified.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

USB devices must pass USB Implementers Forum (IF) tests or be USB-IF certified.

USB-C PD-capable devices, hubs, and chargers must implement version 1.1 of the USB Type-C spec or higher and revision 2.0 v1.1 of the PD spec or higher, and must use PD silicon that is certified according to the USB-IF's USB-C product testing matrix.

For more information, see the white paper on Windows Logo Kit USB-IF Testing:

- <http://www.microsoft.com/whdc/connect/usb/wlk-usb-if-testing.mspix>

Device.Connectivity.UsbDevices.DeviceAttachLessThan100ms

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

Applies to	Windows 11 Client 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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 several 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 11 Client x64 Windows 11 Client ARM64

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 ForceHCRResetOnResume 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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

All USB device requests contain a wLength field. Responses by the USB device to host requests must be of size $\leq$ wLength field of the device request as defined in the USB Specification, Revision 1.1 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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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.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 11 Client x64 Windows 11 Client ARM64

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](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 11 Client x64 Windows 11 Client ARM64

Description

All Wireless USB devices must get a Certified Wireless USB Logo from the USB-IF.

Device.Connectivity.UsbDevices.WirelessUsbWiMediaAlliance

Certified Wireless USB device or host must pass all required WiMedia Alliance compliance tests.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Wireless USB device must pass WiMedia Alliance radio compliance tests.

Device.Connectivity.UsbDevices.Discretionary

USB function controllers must be tested with Microsoft's USB function drivers.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

Systems that have USB function controllers (or USB dual-role controllers in device role) must be tested with Microsoft's USB function drivers installed and enabled.

Device.Connectivity.UsbDevices.USBC

Device.Connectivity.UsbDevices.USBC.USBTypeCAltModeCertification

USB Type-C Alternate Mode devices are certified with their respective organizations.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

If a USB Type-C device supports an Alternate Mode Standard and the industry group that owns that Standard has a corresponding certification for it, the device must use PD silicon that is certified by that industry group in addition to the USB-IF.

For example, if a USB Type-C device's PD silicon supports DisplayPort Alternate Mode in addition to USB and PD, then that silicon must be certified by VESA for the DisplayPort functionality, in addition to the USB-IF for USB and PD functionality.

Certification for vendor-specific Alternate Modes is not required.

Device.Connectivity.UsbDevices.USBC.USBTypeCCertifiedCables

USB Type-C Devices that ship with cables ship with certified cables

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

If a device is USB Type-C and ships with a cable or an adapter, the cable and/or adapter must be USB-IF certified.

In addition, if the USB Type-C cable or adapter is used for an Alternate Mode Standard and the industry group that owns that Standard has a corresponding certification, the cable or adapter must get that certification.

Device.Connectivity.UsbDevices.USBC.USBTypeCAltModeDeviceCompat

USB Type-C Alternate Mode Device Compatibility

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

An Alternate Mode device that only has a single USB port that connects to the downstream facing port (DFP) must also provide USB functionality equivalent to its Alternate Mode.

For example, a Thunderbolt Alternate Mode USB-C storage device must also be able to function as a USB mass storage device over USB 2.0 or USB 3.0.

Note: Alternate Mode adapters are not required to provide USB functionality equivalent to its Alternate Mode. For example, a DisplayPort or MHL Alternate Mode USB-C adapter is not required to also support a video protocol that works over USB 2.0 or USB 3.0.

Device.Connectivity.UsbHub

Requirements that apply only to USB Hubs

Device.Connectivity.UsbHub.IdentifyNumOfUserAccessiblePorts

A USB hub must correctly identify and report the number of ports that the user can access.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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, Section 11.23, 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 11 Client x64 Windows 11 Client ARM64

Description

USB hubs must support the selective suspend state, as stated in both the USB Specification and other logo 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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Connectivity.UsbHub.USBC

Device.Connectivity.UsbHub.USBC.USBTypeCHubCompat

USB Type-C Hub Compatibility

Applies to	Windows 11 Client x64

Description:

USB Type-C hubs must use multiplexers for their SuperSpeed Ports so that a single port can be mapped to either orientation of the attached plug. This enables attaching a USB-C device in either orientation without UI implications or losing state.

If the USB Type-C hub is designed to provide power back to the upstream Host, then its upstream port must be implemented as a charging UFP. In addition, if its upstream port initially resolves as a DFP for data, it must initiate a data role swap; if its upstream port initially resolves as a power sink, it must initiate a power role swap.

Device.Connectivity.WSD

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 11 Client 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 11 Client 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 11 Client 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 11 Client 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 11 Client 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 that 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.WSDDiscovery

Devices Profile for Web Services (DPWS) Devices interacting with the Web Services on Devices API (WSDAPI) implement WS-Discovery

Applies to	Windows 11 Client 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>

Device.DevFund

Device.DevFund.DeviceGuard

All kernel drivers must be built to be compatible with <http://aka.ms/DeviceGuard>

Device.DevFund.DeviceGuard.DriverCompatibility

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](#).

The attributes that must be set on the sys file are covered in Device.DevFund.Security.DriverAttributes requirements section.

Device.DevFund.DriverFramework.KMDF

Driver framework requirements for KMDF

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 11 Client x64 Windows 11 Client ARM64

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:

<https://learn.microsoft.com/windows-hardware/drivers/wdf>

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. To enable Enhanced Verifier, set the following registry values for the driver's Parameters\Wdf key:

```
[HKLM\System\CurrentControlSet\Services\<driver>\Parameters\Wdf]
EnhancedVerifierOptions    REG_DWORD    1
VerifierOn                 REG_DWORD    1
TrackHandles              REG_MULTI_SZ    *
```

- Driver Verifier. To enable Driver Verifier, use the following command:

```
Verifier /standard /driver <sys file name>
```

For more information about Driver Verifier, visit the following website:

<https://learn.microsoft.com/windows-hardware/drivers/devtest/verifier-command-line>

Device.DevFund.DriverFramework.KMDF.WDFProperINF

Windows Driver Framework (WDF) driver INF files must be properly structured.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 WDF section, as follows:

For Kernel-Mode Driver Framework (KMDF) drivers:

```
[DDInstall.Wdf]
KmdfService= <ServiceName>, <Kmdf_Install>
```

```
[Kmdf_Install]
KmdfLibraryVersion= <library version>
```

- All WDF drivers that use a WinUSB driver must register for WinUSB service properly as follows. Refer to the winusb.inf file for other variations, for example registering as a filter driver.

```
[DDInstall]
Include=WINUSB.inf
Needs=WINUSB.NT
```

```
[DDInstall.HW]
Include=WINUSB.inf
Needs=WINUSB.NT.HW
```

```
[DDInstall.Services]
Include=WINUSB.inf
Needs=WINUSB.NT.Services
```

- Service names, hardware IDs (HWIDs), and display names cannot be pasted from WDF samples.

Design Notes:

For more information about WDF-specific INF settings, visit the following websites:

<https://learn.microsoft.com/windows-hardware/drivers/wdf/specifying-wdf-directives-in-inf-files>

Device.DevFund.DriverFramework.UMDF

Driver framework requirements for UMDF

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 11 Client x64 Windows 11 Client ARM64

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\Windows NT\CurrentVersion\WUDF\Services\{193a1820-d9ac-4997-8c55-be817523f6aa}]
TrackObjects    REG_DWORD 1
TrackRefCounts  REG_DWORD 1
```

UMDF drivers must also meet the following requirements:

- UMDF 2 is the latest version of UMDF and supersedes UMDF 1. All new UMDF drivers should be written using UMDF 2.0 or later. Writing UMDF 1.x drivers is not supported.
- 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.

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:

<https://learn.microsoft.com/windows-hardware/drivers/wdf/developing-drivers-with-wdf>

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:

<https://learn.microsoft.com/windows-hardware/drivers/devtest/application-verifier>

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:

<https://learn.microsoft.com/windows-hardware/drivers/wdf>

<https://learn.microsoft.com/windows-hardware/drivers/wdf/getting-started-with-umdf-version-2>

<https://learn.microsoft.com/windows-hardware/drivers/wdf/determining-why-the-reflector-terminated-the-host-process>

Device.DevFund.DriverFramework.UMDF.WDFProperINF

Windows Driver Framework (WDF) driver INF files must be properly structured.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 WDF section, as follows:

For User-Mode Driver Framework (UMDF) drivers:

```
[DDInstall.Wdf]
UmdfService=<ServiceName>,<Umdf_Install>
UmdfServiceOrder= <service name1> [, <service name2> ...]
```

```
[Umdf_Install]
UmdfLibraryVersion= <library version>
ServiceBinary= <binary path>
```

- All UMDF driver INF files must register WUDFRD service properly as follows. Refer to wudfrd.inf for other variations, for example registering as a filter driver.

```
[DDInstall]
```

```

Include=WUDFRD.inf
Needs=WUDFRD.NT

[DDInstall.HW]
Include=WUDFRD.inf
Needs=WUDFRD.NT.HW

[DDInstall.Services]
Include=WUDFRD.inf
Needs=WUDFRD.NT.Services

```

- All WDF drivers that use a WinUSB driver must register WinUSB service properly as follows. Refer to winusb.inf for other variations, for example registering as a filter driver.

```

[DDInstall]
Include=WINUSB.inf
Needs=WINUSB.NT

[DDInstall.HW]
Include=WINUSB.inf
Needs=WINUSB.NT.HW

[DDInstall.Services]
Include=WINUSB.inf
Needs=WINUSB.NT.Services

```

- Service names, hardware IDs (HWIDs), and display names cannot be pasted from WDF samples.

Design Notes:

For more information about WDF-specific INF settings, visit the following websites:

<https://learn.microsoft.com/windows-hardware/drivers/wdf/specifying-wdf-directives-in-inf-files>

<https://learn.microsoft.com/windows-hardware/drivers/wdf/adding-the-reflector>

Device.DevFund.DriverSecurity

Device.DevFund.DriverSecurity.DriverPackage

All binaries must be digitally signed by Microsoft, including all binaries in a compressed file

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Driver packages may not include any 3rd party UI components, apps, or settings. Instead applications from the Windows Store should be provided for user interfaces and may also leverage Settings app extensibility to achieve custom configuration on hardware level tuning or setting.

Additionally, for Windows 11 S only

Drivers must not reference any of the following inbox components:

- bash.exe
- cdb.exe
- cmd.exe
- cscript.exe
- csi.exe
- dnx.exe
- kd.exe
- lxssmanager.dll
- msbuild.exe
- ntsd.exe
- powershell.exe
- powershell_ise.exe
- rcsi.exe
- reg.exe
- regedt32.exe
- windbg.exe
- wmic.exe
- wscript.exe

Device.DevFund.Security.DriverAttributes

Applies to	Windows 11 Client x64 Windows 11 ARM x64

Security and reliability improvements for Windows. These required fields in the driver file improve Microsoft's security teams' ability to reliably revoke drivers with malicious behaviors and security vulnerabilities being exploited by Advanced Persistent Threats (APTs).

The following attributes are required to be set on the driver sys file. The HLK test will verify the existence of the attributes listed below and validate the data format.

Attribute	Description/Format
CompanyName	The name of the company which produced the driver file, for example, "Microsoft Corporation". Ideally, this should match the legal name provided to HDC.
FileDescription	A one-line description of the driver which may be displayed to users. For example, "Keyboard driver for AT-style keyboards".
OriginalFilename	Original name of the file and ".sys" extension, not including the path. For example, "driver.sys". This field is immutable whereas the filename can be modified by the user.
ProductName	Name of the product with which the driver is distributed.
VS_FIXEDFILEINFO:	

FileVersion	Binary version number for the driver file. The version consists of two 32-bit integers, defined by four 16-bit integers. For example, "FILEVERSION 3,10,0,61"
ProductVersion	Binary version number for the product with which the file is distributed. Similar to the FileVersion attribute.

You can find more details on the [StringFileInfo Block](#) and [VERSIONINFO resource on Microsoft Docs](#).

Device.DevFund.Firmware

Driver package requirements for firmware update package

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 11 Client x64 Windows 11 Client ARM64

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.

Device.DevFund.HALExtension

Device.DevFund.HALExtension.HAL

These requirements apply to HAL Extensions that is submitted to MS for approval and signing.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

In addition to standard driver requirements, the following requirements apply to HAL

- HAL Extension should work with system HAL seamlessly.
- HAL Extension should be signed correctly.

Device.DevFund.Memory

Requirements related to memory profile

Device.DevFund.Memory.DriverFootprint

Drivers must occupy a limited memory footprint.

Applies to

Windows 11 Client x64

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 WHLK 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 11 Client 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.

Device.DevFund.Reliability

Reliability tests containing the content of the former DEVFUND tests.

Device.DevFund.Reliability.BasicReliabilityAndPerformance

Drivers are architected to maximize reliability and stability and do not "leak" resources such as memory.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client 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 11 Client x64 Windows 11 Client ARM64

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 Logo 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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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](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 11 Client x64 Windows 11 Client ARM64

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.

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 Lab Kit (WHLK). 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.

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), Version 1.0, 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

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](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).

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 "DF - InfVerif INF Verification" test in the Windows Hardware Lab Kit (WHLK) 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 11 Client 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 11 Client 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 "PnPDTTest 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 11 Client 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.		
<u>Design Notes:</u>		
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.3rdParty

Reliability tests containing content of the former DEVFUND tests.

Device.DevFund.Reliability.3rdParty.FormerTests

Former Tests Mapping Requirement

Applies to		Windows 11 Client 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.	

Device.DevFund.Reliability.Interrupts

Reliability with respect to device interrupts

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 11 Client x64 Windows 11 Client ARM64
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.	

Device.DevFund.ReliabilityDisk

Reliability tests targeting disk devices

Device.DevFund.ReliabilityDisk.IOCompletionCancellation

A device driver must follow the design details in the I/O Completion/Cancellation Guidelines (DEVFUND-0013).

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

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.mspix>.

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.

Device.DevFund.Rollback

Driver Rollback

Device.DevFund.Rollback.Driver

Applies to	Windows 11 Client x64

Description

In order for end users to be able to roll back a driver (i.e. reinstall the previously installed driver), device drivers must be able to be uninstalled and reinstalled with no failures.

Device.DevFund.Security

Additional requirements related to security

Device.DevFund.Security.AntiMalware

Drivers must be devoid of malware.

Applies to	Windows 11 Client x64 Windows 11 ARM x64

Description

Drivers must not include software intentionally designed to cause disruption, leak private information, gain unauthorized access, deprive users access to information or which unknowingly interferes with the user's computer security and privacy.

Design Notes:

During the test, a full-system scan will be performed with Microsoft Defender Antivirus.

Device.Devfund.Security.CodeObfuscation

"Packing" or use of other code obfuscation techniques in drivers is forbidden

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description

Details of the requirement

The use of “packers” and/or code obfuscation techniques makes binary analysis of content anywhere from difficult to impossible. To increase speed, and fidelity of Microsoft analysis to better protect our mutual customers content leveraging obfuscation will be rejected.

Device.DevFund.Security.HardwareEnforcedStackProtection

Capable devices should have Hardware-enforced Stack Protection on, to ensure OS protects code that requests Shadow Stack Protection

Applies to	Windows 11 Client x64

Enhancement Scenarios:

Description:

Security improvements for binaries. Binaries will be protected from return address hijacking via shadow stack mechanism.

Hardware-enforced Stack Protection is Windows 10's implementation of Shadow Stacks. Shadow Stacks is a mechanism to enforce back-edge control flow integrity, by maintaining a secondary stack that shadows the data stack and stores the intended control flow of a program. This prevents an exploit technique known as return-oriented programming (ROP) which hijacks the instruction pointer by overwriting the return address via a memory corruption vulnerability. The feature is described here: <https://techcommunity.microsoft.com/t5/windows-kernel-internals/understanding-hardware-enforced-stack-protection/ba-p/1247815>

Today, binaries that are compiled with /CETCOMPAT linker option marks the binary as Shadow Stack-compatible, by setting a bit in the PE header. This protection is offered to a binary only if the chipset supports Shadow Stacks via new instructions and PTE protections, if the OS supports Shadow Stack, and if the binary requests this protection (via linker flag or in CreateProcess).

Linker flag: <https://docs.microsoft.com/en-us/cpp/build/reference/cetcompat?view=vs-2019>

PE bit: <https://docs.microsoft.com/en-us/windows/win32/debug/pe-format#extended-dll-characteristics>

.

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 11 Client 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.

Device.Devfund.Security.SoftwareBillofMaterials

Partners are required to submit Software Bill of Materials (SBOMs) with their driver submissions to Hardware Dev Center (HDC)

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description

As part of Executive Order 14028, the United States government established a requirement for companies to provide a Software Bill of Materials (SBOM) for their software products. In an effort to streamline SBOM production, and standardize content across the ecosystem. Microsoft has authored tooling, and documentation which is available in the Windows Driver Kit. Submission of an SBOM with your driver, UEFI, or LSA package to Hardware Dev Center for retail trusted signing, will be required. Submissions not containing a SBOM will be rejected.

Federal Register:

<https://www.federalregister.gov/executive-order/14028>

<https://www.federalregister.gov/documents/2021/06/02/2021-11592/software-bill-of-materials-elements-and-considerations>

Additional Information

- WHCP has chosen to support the SPDX standard for content processible by Hardware Dev Center.
- Microsoft does not have plans to distribute, or redistribute SBOMs submitted by WHCP participants at this time.
- Details around support in HDC, with timelines will be published to the HDC SBOM Blog (https://aka.ms/WHCP_SBOM)

Device.DevFund.StaticTools

Device.DevFund.StaticTools.StaticAnalysisForDrivers

Driver development includes static analysis to improve reliability using CodeQL.

Applies to	Windows 11 Client x64

Description

Kernel-mode driver submissions are required to pass the Static Tools Logo Test for certification. CodeQL must be used for static analysis of a kernel-mode driver for submission to Microsoft for signature or certification.

Design Notes:

[CodeQL](#) has been found to be highly effective in improving driver reliability by identifying coding issues that would be otherwise difficult to find. Issues discovered in the list of “Must-Fix” queries must be corrected in driver code for compatibility submission. Please see the [public documentation](#) for details.

In those cases where a single binary is used for Windows Core and Windows Desktop, and/or Windows Server, submissions, and where the Windows Core or Desktop OS version is being certified or signed for a Semi Annual Channel (SAC) release, and the binary is also being certified or signed for a Long Term Servicing Channel release of Server, the process is as follows;

- Use the version of the [Enterprise WDK](#) for the OS you are targeting to generate individual DVLs. This should provide the ability to use the desired kit to build drivers for any specific Client SAC and successfully generate a DVL that can be submitted for one or more versions of Windows Server LTSC certification(s). The resulting DVL file will be required to be copied over to the test machine and picked up by the HLK.

Exceptions

A driver that identifies via its INF as one of the following system-defined [device classes](#) is exempt from running CodeQL for Drivers for Windows 11, 24H2 release certification:

- “display”

How to create

- i. [CodeQL logs](#)
- i. [DVL results file](#)

Device.DevFund.Driver

Additional requirements related to drivers.

Device.DevFund.Driver.Componentized

Device driver must be fully componentized per C requirement of DCH

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All device drivers must be DCH-compliant to be certified starting with Windows11, 21H2. DCH design principles are described in detail [here](#).

This requirement specifies that all device drivers must be properly **componentized**:

Componentized (**C**): Edition-specific, OEM-specific, and optional customizations to the driver are separate from the base driver package. As a result, the base driver, which provides only core device functionality, can be targeted, flighted, and serviced independently from the customizations.

Device.DevFund.Driver.NoGraphicalUI

Driver packages should not include any graphical user interfaces nor require any

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Device driver packages must not include any custom applications or UI. All UI and custom applications should be released through the Windows Store or as a stand alone offering from the driver.

It is acceptable for an application to communicate with the driver to provide additional value add features but the driver must be able to function properly and with core functionality intact without the presence of any applications.

This requirement is the "H" requirement from the [DCH requirements](#) which are required for drivers certifying for Windows 11, 21H2.

Device.DevFund.Driver.Runtime.FilePaths

A driver binary's file accesses must be relative to file handles returned by the operating system or using fully qualified path strings provided by the operating system.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description

Compliance with driver package isolation with regards to runtime file paths means that driver binaries do not access any file locations using hardcoded fully qualified paths.

For additional information about developing isolated driver packages, see the following guidance:

[Driver package isolation - Windows drivers | Microsoft Learn](#)

[Run From Driver Store - Windows drivers | Microsoft Learn](#)

Requirement

Paths for file accesses must be relative to file handles managed by the operating system, as supplied by supported driver model and framework APIs. Paths under "\DriverData" may also be used in the manner described in the driver package isolation documentation.

Exceptions

Current exceptions to this requirement are paths that start with "\SystemRoot\System32". Exceptions will be removed in a future WHCP release.

Device.DevFund.Driver.Runtime.RegistryPaths

Driver binaries must only access registry keys relative to handles provided by the OS.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description

Compliance with driver package isolation requirements means that driver binaries do not access any registry locations using fully qualified absolute paths (e.g. HKLM\...) at any point during its runtime execution.

For additional information about developing isolated drivers, see the following guidance:

[Driver package isolation - Windows drivers | Microsoft Learn](#)

Requirement

Registry access must only be performed on registry locations relative to registry handles managed by Plug and Play (PnP), as supplied by supported driver model and framework APIs. In addition, driver binaries may read from, but not write to HKLM\Hardware.

Exceptions

Current exceptions to this requirement can be found in section [Device.DevFund.INF.RegistryPaths](#) in the table labeled "Registry Path Exceptions".

Deprecated Runtime Registry Paths Exceptions

Current deprecated runtime registry path exceptions can be found in section [Device.DevFund.INF.RegistryPaths](#) in the table labeled "Deprecated Registry Paths Exceptions".

Device.DevFund.INF

Device.DevFund.INF.Componentized

INF must be fully componentized per C requirement of DCH

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

All device drivers must be DCH-compliant to be certified starting with Windows 11, 21H2. DCH design principles are described in detail [here](#).

This requirement specifies that all INF files must be properly **componentized**:

Componentized (**C**): Edition-specific, OEM-specific, and optional customizations to the driver are separate from the base driver package. As a result, the base driver, which provides only core device functionality, can be targeted, flighted, and serviced independently from the customizations.

Device.DevFund.INF.Declarative

When using an INF, it must fully describe the driver installation.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Driver installation must be fully described by the INF file. That means all registry, services, files, etc. necessary for the driver to function completely must be included and described by INF directives. Below are some guidelines to follow and a set of sections and directives that cannot be used.

- Driver installation and removal must use Windows-based methods. Therefore, only INF-based installation routines are allowed. A device driver must have a properly formatted INF for its device.

- An INF file may not specify any co-installers within a DDInstall.Coinstallers section. A co-installer typically writes additional configuration information or performs other installation tasks.
- An INF file may not specify a custom class installer or class co-installer within a ClassInstall32 section. Therefore, a driver package cannot execute a custom class installer during device installation.
- 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.
- INF files cannot 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, is not supported.
- INF files cannot be used to perform device resource configuration or non-PnP related configuration tasks.
- INF files cannot use directives that delete registry entries, services, or files.
- INF files cannot use directives that perform operations on existing state, such as renaming files or appending to registry entries.
- 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 SuiteMask flag.

The following INF sections and directives may not be referenced in an INF file:

- RegisterDlls
- UnregisterDlls
- ProfileItems
- UpdateIni
- UpdateIniFields
- Ini2Reg
- ExcludeID
- [DDInstall.LogConfigOverride] section
- LogConfig
- [DDInstall.FactDef] section
- DelReg (Exception: Allowed for HKR-based values)
- DelService
- DelProperty
- BitReg
- DelFiles
- RenFiles
- [DDInstall.Coinstallers] section

All device drivers must be DCH-compliant to be certified starting with Windows 11, 21H2 (unless otherwise granted an exception). DCH design principles are described in detail [here](#). This requirement represents the “D” requirement of “DCH”.

[InfVerif /k](#) will be used to enforce this requirement.

Exceptions	If the device uses an inbox driver and INF (and, therefore, does not provide an INF file), this requirement does not apply.

Device.DevFund.INF.FilePaths

INF files can only copy files to DIRID 13.

Applies to**Windows 11 Client x64**
Windows 11 Client ARM64
Windows Server x64

Description

INF files cannot copy any files to a global destination location, and instead must make all files in the driver package be 'Run from Driver Store'. All isolated driver packages leave their driver package files in the driver store.

Supporting driver files maintained in the Driver Store, such as libraries and data files, must be dynamically discovered at runtime by the drivers and services/apps instead of relying on hardcoded file paths. More details can be found [here](#):

[Run From Driver Store - Windows drivers | Microsoft Learn](#)

For additional information on the INF file paths, see the following guidance:

[Driver package isolation - Windows drivers | Microsoft Learn](#)

[Using Dirids - Windows drivers | Microsoft Learn](#)

The following tool can be used to validate compliance:

- Running [InfVerif tool](#) – enforced by InfVerif /h

Requirement

DIRID 13 must be specified in the INF as the destination location for all entries in the [DestinationDirs] section.

Exceptions

Below are exceptions to the requirement, these exceptions will be removed in a future WHCP release.

DIRID Value	File Root Directory	File Subdirectory Path
DIRID 10	Windows	
DIRID 11	Windows\System32	
DIRID 12	Windows\System32\drivers	
DIRID 23	Windows\System32\spool\drivers\color	
DIRID 51	Windows\System32\spool	
DIRID 52	Windows\System32\spool\drivers\...	
DIRID 55	Windows\System32\spool\prtprocs\...	
DIRID 16422	Program Files	
DIRID 16425	Windows\SysWOW64	
DIRID 16426	Program Files (x86)	
DIRID 16427	Program Files\Common Files	
DIRID 16428	Program Files (x86)\Common Files	
DIRID 66000	Windows\System32\spool\drivers\...\3	
DIRID 66001	Windows\System32\spool\prtprocs\...	
DIRID 66002	Windows	
DIRID 66003	Windows\System32\spool\drivers\color	
DIRID 66004	Windows\web\printers\...	

Device.DevFund.INF.RegistryPaths

INF files can only write to the Plug and Play (PnP) managed relative registry root.

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Windows Server x64

Description

HKR-based registry settings supplied by an INF can be subsequently accessed by the driver at runtime using the supported Plug and Play (PnP) driver model and framework APIs, as mentioned in

[Device.DevFund.Driver.Runtime.RegistryPaths](#)

For additional information on the INF registry accesses, see the following guidance:

[Driver package isolation - Windows drivers | Microsoft Learn](#)

[INF AddReg directive - Windows drivers | Microsoft Learn](#)

[Porting an INF to follow driver package isolation - Windows drivers | Microsoft Learn](#)

The following tools can be used to validate compliance:

- Running [InfVerif tool](#) – enforced by InfVerif /h

Requirement

INF files must not modify any global registry locations, and instead only use the HKR registry root to modify or create registry information with an AddReg directive.

Exceptions

Below are exceptions to this requirement, these exceptions will be removed in a future WHCP release.

Registry Path Exception List:

Registry Root	Registry Subkey Path
HKLM	SYSTEM\CurrentControlSet
HKLM	SOFTWARE\Classes
HKLM	SOFTWARE\Khronos
HKLM	SOFTWARE\Microsoft\Analog\Providers
HKLM	SOFTWARE\Microsoft\Cellular\MVSettings\DeviceSpecific\CellUX
HKLM	SOFTWARE\Microsoft\Cryptography\Calais\Readers
HKLM	SOFTWARE\Microsoft\Cryptography\Calais\SmartCards
HKLM	SOFTWARE\Microsoft\Cryptography\DRM_RNG
HKLM	SOFTWARE\Microsoft\EAPOL
HKLM	SOFTWARE\Microsoft\Palm\DelayManipulationDuration
HKLM	SOFTWARE\Microsoft\Shell\OEM\QuickActions\ColorProfileQuickAction
HKLM	SOFTWARE\Microsoft\Speech_OneCore\AudioInput

HKLM	SOFTWARE\Microsoft\Windows Media Foundation
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\drivers.desc
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\Drivers32
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\ICM
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\OpenGLDrivers
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify\ScCertProp
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Audio
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Control Panel
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Controls Folder
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Reliability\UserDefined
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
HKLM	SOFTWARE\Wow6432Node\Microsoft\Windows Media Foundation
HKLM	SOFTWARE\Wow6432Node\Khronos
HKLM	SOFTWARE\WowAA32Node\Microsoft\Windows Media Foundation
HKLM	SOFTWARE\WowAA32Node\Khronos
HKCR	

Deprecated Registry Paths Exceptions:

INF files must not modify any global registry locations with an AddReg directive and instead only use the HKR registry root. The following registry paths have been deprecated due to improved alternatives being available. See links under additional information for alternatives.

Registry Root	Registry Subkey Path	Release Registry Path was Deprecated	Additional Information
HKLM	SOFTWARE\Microsoft\Windows\CurrentVersion\PrecisionTouchPad		Link
HKCR	AudioEngine\AudioprocessingObjects		Link
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\WINEVT\Channels		Link
HKLM	SOFTWARE\Microsoft\Windows NT\CurrentVersion\WINEVT\Publishers		Link

Device.DevFund.INF.State

INF files must follow global state design concepts listed in this section.

Applies to	Windows 11 Client x64
	Windows 11 Client ARM64
	Windows Server x64

Description

INF files must not have conflicting global state, such as copying different settings to the same registry location, different files to the same file destination, or different settings for the same service.

For additional information on related INF directives, see the following guidance:

[INF CopyFiles directive - Windows drivers | Microsoft Learn](#)

[INF AddReg directive - Windows drivers | Microsoft Learn](#)

[INF AddService directive - Windows drivers | Microsoft Learn](#)

The following tool can be used to validate compliance:

- Running [InfVerif tool](#) – enforced by InfVerif /h

Requirement

All driver package state must be deterministic and not depend on runtime system attributes. For detailed information see the following:

[InfVerif Error 1330 - Windows drivers | Microsoft Learn](#)

Device.DevFund.INF.Syntax

In addition to existing INF syntax requirements, INF files must now include the following syntax.

Applies to	Windows 11 Client x64
	Windows 11 Client ARM64
	Windows Server x64

Description

In addition to existing INF syntax requirements found in the section labeled “Device.DevFund.INF.Declarative,” INF files must now include the following:

PnpLockdown - 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 file.

For additional information on these requirements, see the following guidance:

[INF Version section - Windows drivers | Microsoft Learn](#)

The following tool can be used to validate compliance:

- Running [InfVerif tool](#) – enforced by InfVerif /h

Requirement

[Version] section must contain the following:

Device.Display.Monitor

Device.Display.Monitor.Base

Base requirements for displays must ensure a good end user experience.

Applies to	Windows 11 Client 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 11 Client 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 11 Client 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 11 Client x64 Windows Server 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).

NOTE: In Windows 10, no minimum refresh rate is required. For native display resolution, the recommended minimum is 60 Hz progressive or the closest frequency appropriate for the region. Lower refresh rates might result in a poor user experience.

For an integrated panel:

The native resolution of the panel must be greater than or equal to 1024 x 768.

For HD15, DVI, HDMI, and DisplayPort connector:

The native resolution of the panel must be greater than or equal to 1024 x 768.

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 11 Client 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.

Device.Graphics

Device.Graphics.AdapterBase

The base feature set required by all graphic devices.

Device.Graphics.AdapterBase.ApplicationVerifier

Graphics driver components must comply with the Application Verifier test criteria.

Applies to	Windows 11 Client 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 files for a graphics adapter or chipset have properly formatted file versions.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The driver information file (.inf), kernel mode driver (.sys), and user mode driver (.dll) file version info must match. In addition, version info for any files identified in the [SignatureAttributes] section of the .inf as PETrust binaries must match the .inf. It is recommended that file version info for additional binaries in a driver package match the .inf.

To be consistent with the prevailing file versioning requirements for legacy Operating systems, file version formatting must follow an AA.BB.CCCCC.DDDDD pattern where:

- AA indicates the Driver model version of the most capable device listed in the .inf
- BB (for WDDM 2.8 drivers and higher)
 - Unused, set to 0.
- BB (for WDDM 1.2 through WDDM 2.7 drivers)
 - indicates highest available D3D Feature Level of the most capable device listed in the .inf
- BB (for WDDM v1.1 drivers and lower)
 - indicates highest available DDI version supported by the most capable device listed in the .inf
- CCCCC is a number up to 5 digits from 0 to 65535 chosen by the vendor

- DDDDD is a number up to 5 digits from 0 to 65535 chosen by the vendor

With each OS coding milestone (i.e. semester), we implement new DDI and WDK impacting features. We will generally increment the WDDM version with each semester, independent of which OS products that feature work ships in. The amount of the increment is generally 0.1 but may vary, for example to indicate a more significant update.

Values for AA field

Driver Model	AA value	OS milestone (semester)
WDDM v2.8	28	Manganese
WDDM v2.7	27	Vibranium
WDDM v2.6	26	19H1
WDDM v2.5	25	RS5
WDDM v2.4	24	RS4
WDDM v2.3	23	RS3
WDDM v2.2	22	RS2
WDDM v2.1	21	RS1
WDDM v2.0	20	TH1
WDDM v1.3	10	
WDDM v1.2	9	
WDDM v1.1	8	
WDDM v1.0	7	
XDDM	6	

Values for BB field (WDDM 2.8 and above):

Unused, set to 0.

Values for BB field (WDDM v1.2 through v2.7):

DirectX Feature Level	BB value
12_x	21
12_1	20
12_0	19
11_1	18
11_0	17

10_1	16
10_0	15
9_3	14
9_2	14
9_1	14

Values for BB field (WDDMv1.1 and earlier):

DDI version	BB value
D3D11-DDI on Feature Level 11_0	17
D3D11-DDI on Feature Level 10	16
D3D10-DDI	15
D3D9 DDI	14

For example:

NOTE: There is no requirement to pad numbers with leading zeros, i.e. 123 does not need to be represented as 00123 for the CCCCC or DDDDD fields. In previous versions of the Windows OS the last two fields were 4 digits, i.e. CCCC.DDDD. Therefore the examples for driver versions prior to Windows 10 and WDDM v2.0 only have 4 digits.

- Windows Vista WDDM v1.0:
 - D3D9 DDI drivers can use 7.14.0000.0000 to 7.14.9999.9999
 - D3D10 DDI drivers can use 7.15.0000.0000 to 7.15.9999.9999
- Windows 7 WDDM v1.1:
 - D3D9 DDI drivers can use 8.14.0000.0000 to 8.14.9999.9999
 - D3D10 DDI drivers can use 8.15.0000.0000 to 8.15.9999.9999
 - D3D11 DDI with FL_10_0 drivers can use 8.16.0000.0000 to 8.16.9999.9999
 - D3D11 DDI with FL_11_0 drivers can use 8.17.0000.0000 to 8.17.9999.9999
- Windows 8 WDDM v1.2 driver:
 - FL_10_0 HW can use 9.15.0000.0000 to 9.15.9999.9999
 - FL_10_1 HW can use 9.16.0000.0000 to 9.16.9999.9999
 - FL_11_0 HW can use 9.17.0000.0000 to 9.17.9999.9999
 - FL_11_1 HW can use 9.18.0000.0000 to 9.18.9999.9999
- Windows 8.1 WDDM v1.3 driver:
 - FL_10_0 HW can use 10.15.0000.0000 to 10.15.9999.9999
 - FL_10_1 HW can use 10.16.0000.0000 to 10.16.9999.9999
 - FL_11_0 HW can use 10.17.0000.0000 to 10.17.9999.9999
 - FL_11_1 HW can use 10.18.0000.0000 to 10.18.9999.9999
- Windows 10 WDDM v2.0 driver:
 - FL_11_1 HW can use 20.18.0000.0000 to 20.18. 65535.65535
 - FL_12_0 HW can use 20.19.0000.0000 to 20.19. 65535.65535
 - FL_12_1 HW can use 20.20.0000.0000 to 20.20. 65535.65535

Enforcement:

A mandatory test in the HLK certification playlist for Windows 10 builds higher than 10586 will enforce the rules above. The test will be optional for older OS versions. For Windows 10 builds after 10586 the WDDM version has been updated to 2.1. Another way to view this is that the mandatory requirement only applies to drivers that are built for WDDM 2.1 or higher.

Device.Graphics.AdapterBase.PowerManagementCompliance

A graphics adapter must comply with VESA and ACPI power management specifications to enable system sleep.

Applies to	Windows 11 Client 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.RunFromDriverStore

The driver files for a graphics adapter or chipset are loaded from the Driver Store.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Drivers must be written so that their components (User Mode Driver, Kernel Mode Driver, etc.) can run directly from the Driver Store. They must not have any dependencies on system paths such as Windows\System32. This includes selecting the *Run from Driver Store* option in the driver information file (.inf).

Note that once this change is made, unique names or absolute paths must be used to load dependencies. For drivers choosing to implement the additional optional side-by-side capability, service names must also be unique per driver version or class of hardware.

Business Justification:

Having drivers run from the Driver Store will improve the upgrade experience. It also enables vendors to support additional "side by side" functionality if they so choose – for example, this would enable the scenario of having two or more GPUs from different families in the same machine running different driver versions.

An additional benefit is that it reduces the disk footprint because drivers must exist in the Driver Store anyway and having an additional copy in Windows\System32 results in duplication.

Enforcement:

The settings in the driver information file (.inf) will be verified by an HLK test for Windows 10 builds higher than 10586.

Device.Graphics.AdapterBase.SubsystemResettable

A graphics subsystem must be resettable to a normal operational state.

Applies to	Windows 11 Client 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.

Device.Graphics.AdapterBase.Symbols

Symbols must be submitted along with the driver for certification

Applies to	Windows 11 Client x64

Description

When graphics drivers are submitted for certification, matching symbol files must be submitted at the same time via the WHKL "Add symbols" workflow. Microsoft uses matching symbols internally for failure analysis. Without these symbols, it is unlikely Microsoft will be able to analyze failures and assist partners in diagnosing driver issues. Symbols for the following files must be included:

- .sys files
- .dll files listed in the [SignatureAttributes] section of the driver .inf as "PETrust" binaries

Symbols may be "public" or "private" per [MSDN guidance](#).

Device.Graphics.AdapterCore

The core feature set applicable to multiple adapter types, such as GPUs, NPUs, Compute Accelerators and Media Accelerators.

Device.Graphics.AdapterCore.DXCoreAttributes

Drivers are required to report DXCore attributes via driver information files (.inf) to support adapter enumeration via the DXCore API.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description:

DXCore attributes are GUIDs that indicate adapter information such as DirectX runtime support and hardware type. DXCore uses these attributes to provide adapter enumeration for applications. For how to properly report DXCore attributes, please refer to the DXCore specification.

Device.Graphics.AdapterCore.D3D12.Metacommands.MachineLearning

IHVs can optionally expose hardware accelerated versions of specific known algorithms to applications utilizing Windows ML and DirectML.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description:

IHVs can expose hardware accelerated versions of specific known algorithms to applications utilizing Windows ML and DirectML for machine learning scenarios by exposing DirectML meta commands. The implementable set of DirectML meta commands are detailed in the "[Direct ML Meta Commands Definitions](#)" document. This document is available to hardware vendors by request to askwindowssl@microsoft.com.

Device.Graphics.AdapterRender

The Render feature of a graphics device.

Device.Graphics.AdapterRender.MinimumDirectXLevel

A graphics adapter must implement the minimum hardware acceleration capabilities.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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/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](http://msdn.microsoft.com/en-us/library/ff471324(v=VS.85).aspx)

NOTE: For Windows Server, graphics drivers are optional. Display Only graphics drivers do not render so they do not have a minimum DirectX requirement. If a Full Graphics driver is loaded on a server, then it must conform to the minimum DirectX feature level.

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 11 Client 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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.AdapterRender.D3D101Core

D3D 10.1 core feature

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.AdapterRender.D3D101WDDM11

D3D 10.1 core feature with WDDM 1.1 additions

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.AdapterRender.D3D101WDDM12

D3D 10.1 core feature with WDDM 1.2 additions

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 11 Client x64
Windows 11 Client ARM64

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.

Device.Graphics.AdapterRender.D3D10ComputeShader

D3D10 Shader Model 4_* Compute Shader Functionality*

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 11 Client x64 Windows 11 Client ARM64

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](#).

Device.Graphics.AdapterRender.D3D10Core

D3D 10 core feature

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 11 Client x64 Windows 11 Client ARM64

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

Device.Graphics.AdapterRender.D3D10D3D11LogicOps

D3D10-D3D11 Logic Ops

Device.Graphics.AdapterRender.D3D10D3D11LogicOps.D3D10CoreD

If the graphics hardware implements Logic Ops functionality, it must conform to the D3D11 hardware specifications.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](#).

Device.Graphics.AdapterRender.D3D10Multisampling4X

D3D10 Multisampling (4X)

Device.Graphics.AdapterRender.D3D10Multisampling4X.D3D10CoreA

If the graphics hardware implements D3D10 4x Multisampling, it must conform to the D3D10 hardware specifications.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](#).

Device.Graphics.AdapterRender.D3D10Multisampling8X

D3D10 Multisampling (8X)*

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 11 Client x64 Windows 11 Client ARM64

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](#).

Device.Graphics.AdapterRender.D3D10WDDM11

D3D 10 core feature with WDDM 1.1 additions

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 11 Client x64****Windows 11 Client ARM64**

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

Device.Graphics.AdapterRender.D3D10WDDM12

D3D 10 core feature with WDDM 1.2 additions

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 11 Client x64****Windows 11 Client ARM64**

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 11 Client x64 Windows 11 Client ARM64

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 <i>any</i> ArraySize (including > 2)

GDI interop	Restricted to ArraySize = 1	Generalized to <i>any</i> ArraySize (including > 2)
HLSL non-arrayed sampler declarations	Restricted to ArraySize = 1	Generalized to <i>any</i> 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.

Device.Graphics.AdapterRender.D3D11Core

D3D 11.1 core feature

Device.Graphics.AdapterRender.D3D11Core.D3D11CorePrimary

If a graphics device supports Direct3D 11.1, it must comply with the Direct3D 11.1 and DXGI Specifications.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.AdapterRender.D3D11ASTC

Device.Graphics.AdapterRender.D3D11ASTC.CoreRequirement

Applies to	Windows 11 Client 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

Device.Graphics.AdapterRender.D3D11ConservativeRasterization

Device.Graphics.AdapterRender.D3D11ConservativeRasterization.CoreRequirement

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

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.

Device.Graphics.AdapterRender.D3D11Core

D3D 11 core feature

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 11 Client x64 Windows 11 Client ARM64

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 Direct 3D 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.

Device.Graphics.AdapterRender.D3D11DoublePrecisionShader

D3D11 Double Precision Shader Functionality*

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 11 Client x64 Windows 11 Client ARM64

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](#).

Device.Graphics.AdapterRender.D3D11DriverCommandLists

D3D11 Driver Command Lists*

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 11 Client x64 Windows 11 Client ARM64

Description

The D3D11 Specification allows for optionally implementing DriverCommandList 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.

Device.Graphics.AdapterRender.D3D11DriverConcurrentObjectCreation

D3D11 Driver Concurrent Object Creation*

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.AdapterRender.D3D11Level9WDDM13

WDDM1.3 updates to the D3D9 UM DDI

Device.Graphics.AdapterRender.D3D11Level9WDDM13.Level9Instancing

Level 9 Instancing

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.D3D11PartialPrecision

D3D11 Partial Precision shader support

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 11 Client x64 Windows 11 Client ARM64

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](#).

Device.Graphics.AdapterRender.D3D11RasterizerOrderedViews

Device.Graphics.AdapterRender.D3D11RasterizerOrderedViews.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](#).

Device.Graphics.AdapterRender.D3D11StencilReference

Device.Graphics.AdapterRender.D3D11StencilReference.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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](#).

Device.Graphics.AdapterRender.D3D11TypedUAVLoads

Device.Graphics.AdapterRender.D3D11TypedUAVLoad.CoreRequirement

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

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](#)

Device.Graphics.AdapterRender.D3D11WDDM12

D3D 11 core feature with WDDM 1.2 additions

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 11 Client x64****Windows 11 Client ARM64**

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 Direct 3D 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.

Device.Graphics.AdapterRender.D3D11WDDM12DoublePrecisionShader

D3D11 Double Precision Shader Functionality with additional ops codes introduced with WDDM 1.2*

Page **172** of **561**

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 11 Client x64****Windows 11 Client ARM64**

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](#).

Device.Graphics.AdapterRender.D3D11WDDM21

WDDM 2.1 is an optional version level introduced for Windows 10 builds greater than 10586. If a driver implements WDDM 2.1 then it must implement all of the WDDM 2.1 Requirements. WDDM 2.0 drivers will continue to run on the newer Windows 10 builds.

Device.Graphics.AdapterRender.D3D11WDDM21.CoreRequirement

Applies to**Windows 11 Client x64**

Terms: If-Implemented

Description:

1. Extended NV12/ 420 Shared Texture Support

Note: This feature is Mandatory for a WDDM 2.1 driver supporting D3D11.

The driver must support the sharing of a few texture formats with more properties, in order to enable frame server scenarios. The goal is zero-copy between capture devices and video engines.

2. eFSE/UWP Optimization Support

Note: This feature is Mandatory for a WDDM 2.1 driver supporting D3D11 or D3D9.

The driver must support the new `pfnAcquireResource/pfnReleaseResource` methods intended to enable driver optimizations for eFSE (emulated Full Screen Exclusive) / UWP (Universal Windows Platform) performance parity with FSE.

Device.Graphics.AdapterRender.D3D12ASTC

Device.Graphics.AdapterRender.D3D12ASTC.CoreRequirement

Applies to

Windows 11 Client x64

Terms: If-Implemented

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 equivilovants.)

- 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.

Device.Graphics.AdapterRender.D3D12.AlphaBlendFactor

WDDM 3.1 drivers must support the alpha-constant blend factor for blend operations.

Device.Graphics.AdapterRender.D3D12.AlphaBlendFactor.CoreRequirement

D3D12 devices expose alpha blend factor support.

Applies to	Windows 11 Client x64

Description

This feature adds D3D12_BLEND_ALPHA_FACTOR and D3D12_BLEND_INV_ALPHA_FACTOR to D3D blend types.

AlphaBlendFactor: https://dev.azure.com/cga-exchange/_git/docs? a=preview&path=/d3d/VulkanOn12.md&version=GBmaster&fullScreen=true&anchor=alpha-blend-factor

Device.Graphics.AdapterRender.D3D12ConservativeRasterization

Device.Graphics.AdapterRender.D3D12ConservativeRasterization.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Graphics.AdapterRender.D3D12Core

D3D 12 core feature

Device.Graphics.AdapterRender.D3D12Core.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description:

Display drivers for Direct3D12 must meet all requirements defined in the **Direct3D 12 Specifications**.

1. 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.

2. 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.

3. 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
- A D3D12 driver must be able to sample a beginning of pipeline stamp in addition to an end of pipeline timestamp. The time on the beginning of pipeline timestamp must be from when the GPU begins execution of a correlated graphics command on its processing units it uses to execute the command.
- 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.

4. 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.

5. Indirect Rendering:

- A D3D12 driver must implement GPU drawing/dispatch from indirect argument buffers.
- Rendering with an arbitrary 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.
- Arbitrary 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.

6. 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.

Device.Graphics.D3D12.DirectSRMetacommands

All graphics drivers supporting DirectSR metacommands must pass DirectSR metacommand tests.

Device.Graphics.D3D12.DirectSRMetacommands.Base

DirectSR metacommands are used to support native super resolution upscaling on a GPU via the DirectSR APIs.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows Server x64

Description

DirectSR uses metacommands to initialize and submit super resolution workloads to GPUs. While there are no prescriptive algorithms for upscaling techniques, drivers must properly support the DirectSR metacommands as detailed in the DirectSR specification.

Device.Graphics.AdapterRender.D3D12.InvertedViewports

WDDM 3.1 drivers must support setting a viewport's MaxDepth as less than its MinDepth and support setting its height as negative.

Device.Graphics.AdapterRender.D3D12.InvertedViewportDepthFlipsZ.CoreRequirement

D3D12 devices expose support for "inverted" depth viewports.

Applies to	Windows 11 Client x64

Description

This feature relaxes the requirement in D3D that the viewport MinDepth < MaxDepth.

Inverted viewports: https://dev.azure.com/cga-exchange/_git/docs? a=preview&path=/d3d/VulkanOn12.md&version=GBmaster&fullScreen=true&anchor=inverted-viewports

Device.Graphics.AdapterRender.D3D12.InvertedViewportHeightFlipsY.CoreRequirement

D3D12 devices expose support for inverted Y viewports.

Applies to	Windows 11 Client x64

Description

This feature allows setting D3D's viewport height to a negative value to flip the Y axis orientation.

Inverted viewports: https://dev.azure.com/cga-exchange/_git/docs? a=preview&path=/d3d/VulkanOn12.md&version=GBmaster&fullScreen=true&anchor=inverted-viewports

Device.Graphics.AdapterRender.D3D12.GPUUploadHeaps

Device.Graphics.AdapterRender.D3D12.GPUUploadHeaps.CoreRequirement

D3D12 graphics hardware supporting GPU Upload Heaps must meet the specified requirements.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64 Windows 11 Server x64

Description

GPU upload heaps allow developers to make use of resizable bar, which was previously not allowed in D3D. GPU upload heaps also offers developers a way to directly upload resources to a heap that they will use later, this can save copy operations. Some games will benefit from this feature to gain a big performance boost.

For a link to a specification fully describing the feature, see: https://dev.azure.com/cga-exchange/docs/_git/docs?version=GBmaster&path=/d3d/D3D12GPUUploadHeap.md

Device.Graphics.AdapterRender.D3D12Multiadapter

Device.Graphics.AdapterRender.D3D12Multiadapter.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Graphics.AdapterRender.D3D12.NonNormalizedSampling

Device.Graphics.AdapterRender.D3D12.NonNormalizedSampling.CoreRequirement

D3D12 graphics hardware supporting non normalized sampling coordinates must meet the specified requirements.

Applies to	Windows 11 Client x64

Description

Vulkan allows sampling from textures using non-normalized coordinates, as specified during sampler creation. To enable this in D3D12, DDI 0100 adds a sampler flag indicating to use non-normalized coordinates.

Samplers with D3D12_SAMPLER_FLAG_NON_NORMALIZED_COORDINATES set must sample from the target texture at the coordinates specified, unscaled by the texture's dimensions.

When non-normalized coordinates are used, the following restrictions apply:

- The SRV being sampled must be D3D12_SRV_DIMENSION_TEXTURE1D or D3D12_SRV_DIMENSION_TEXTURE2D
- The SRV's mip-count must be 1.
- The sampling coordinate valid range is x:[0, width) and y:[0, height)
- The Offset parameter in Sample* functions must not be used
- The D3D12_FILTER must be one of
 - D3D12_FILTER_MIN_MAG_MIP_POINT
 - D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT
 - D3D12_FILTER_MINIMUM_MIN_MAG_MIP_POINT
 - D3D12_FILTER_MINIMUM_MIN_MAG_LINEAR_MIP_POINT
 - D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_POINT

- D3D12_FILTER_MAXIMUM_MIN_MAG_LINEAR_MIP_POINT

Where the min and mag filters are equal, mip is POINT, and comparison and anisotropy are disabled.

- MinLOD and MaxLOD must be 0
- AddressU and AddressV must be D3D12_TEXTURE_ADDRESS_MODE_CLAMP or D3D12_TEXTURE_ADDRESS_MODE_BORDER

The result of sampling outside of the valid range is generally undefined in terms of what value may be returned, but additionally undefined in terms of what state the device is left in if AddressU or AddressV are not appropriately set.

Support for non-normalized texture sampling is implied by, and must be supported on, drivers that report their DDI version to be 0100 or greater.

Device.Graphics.AdapterRender.D3D12RasterizerOrderedViews

Device.Graphics.AdapterRender.D3D12RasterizerOrderedViews.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Graphics.AdapterRender.D3D12StencilReference

Device.Graphics.AdapterRender.D3D12StencilReference.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Graphics.AdapterRender.D3D12TypedUAVLoads

Device.Graphics.AdapterRender.D3D12TypedUAVLoad.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Graphics.AdapterRender.D3D12.FormatListCasting

Device.Graphics.AdapterRender.D3D12.FormatListCasting.CoreRequirement

D3D12 graphics hardware supporting Format List Casting must meet the specified requirements.

Applies to	Windows 11 Client x64

Description

Format list casting allows developers to supply a list of possible target castable formats at resource creation. The API is part of, and therefore has a dependency on, Device.Graphics.AdapterRender.D3D12.EnhancedBarriers.

Vulkan allows reinterpretation-casting of resource formats by supplying a target list of castable formats during resource creation. Except for planar and compressed formats, the target formats in Vulkan can be any format whose

block size and texels per block matches the resource's format's. Prior to this feature, even with relaxed format casting rules, D3D12 only allowed casting between formats within their respective format families and only if the original format was typeless.

With this feature, D3D12 provides an API at resource creation to specify a list of castable formats for the resource.

The list

- MAY be empty
- MAY contain uncompressed formats whose unit size matches the resource's format's unit size.
- MAY NOT contain planar formats.
- If the resource's format is non-compressed, then the list may ONLY contain non-compressed formats.

Additionally, if the resource's format is compressed, then a view that casts to a non-compressed format must have MipLevels and ArraySize equal to 1.

Formats not specified in the list during creation MAY NOT be used as view target formats for the resource.

Formats "casted" in this way are *reinterpreted* as the target format for consumption in shaders.

Refer to [CreateCommittedResource3](#), [CreatePlacedResource2](#), and [CreateReservedResource2](#) in Enhanced Barriers for documentation on the new API fields, NumCastableFormats and pCastableFormats used to specify the list.

DDI

Driver support for format list casting is reported via RelaxedFormatCastingSupported in D3D12DDI_OPTIONS_DATA_0090 when GetCaps is called with D3D12DDICAPS_TYPE_OPTIONS_0090; available starting with DDI version e_DDI_12_8_0090.

Device.Graphics.AdapterRender.D3D12.EnhancedBarriers

Device.Graphics.AdapterRender.D3D12.EnhancedBarriers.CoreRequirement

D3D12 graphics hardware supporting Enhanced Barriers must meet the specified requirements.

Applies to	Windows 11 Client x64

Description

Enhanced Barriers give developers independent control over GPU work synchronization, texture layout transitions, and cache flushing ("resource memory access"). Enhanced barriers replace resource state barriers with more expressive barrier types. Enhanced barriers offer less latency, support for concurrent read+write (including self-copy), diverse aliasing topologies, and better concurrency.

Device.Graphics.AdapterRender.D3D12.IndependentStencilRefsAndMasks

Device.Graphics.AdapterRender.D3D12.IndependentStencilRefsAndMasks.CoreRequirement

D3D12 graphics hardware supporting Independent Stencil Refs and Masks must meet the specified requirements.

Applies to	Windows 11 Client x64

Description

Vulkan supports assigning reference and mask values independently for front-face and back-face stencil operations. With this feature, D3D12 adds additional fields for back-face read and write masks to the stencil description, as well as a CommandList function for setting the reference values independently.

DDI

Support for independent front-face and back-face reference and masks will be implied by, and must be supported on, drivers that report their DDI version to be e_DDI_12_8_0095 or greater.

During PSO creation, D3D12DDI_DEPTH_STENCIL_DESC_0095 provides fields for back-face read and write masks, as well as renaming 0025 mask values to be specific to front-face stencil operations.

Device.Graphics.AdapterRender.D3D12.TriangleFan

Device.Graphics.AdapterRender.D3D12.TriangleFan.CoreRequirement

D3D12 graphics hardware supporting triangle fans must meet the specified requirements.

Applies to	Windows 11 Client x64

Description

Vulkan supports the triangle fan primitive topology and software emulation for it is expensive, so D3D will bring back support for it natively.

DDI

DDI version 0097 will revive D3D12DDI_PRIMITIVE_TOPOLOGY_TRIANGLEFAN in the slot reserved for it in D3D12DDI_PRIMITIVE_TOPOLOGY, 6. Triangle fans are sufficiently described on msdn: [Triangle Fans \(Direct3D 9\)](#). Support for the triangle fan primitive topology will be implied by, and must be supported on, drivers that report their DDI version to be e_DDI_12_8_0097 or greater.

Device.Graphics.AdapterRender.D3D12.DynamicPipelineState

Device.Graphics.AdapterRender.D3D12.DynamicPipelineState.CoreRequirement

D3D12 graphics hardware supporting dynamic pipeline state must meet the specified requirements.

Applies to	Windows 11 Client x64

Description

Depth bias state

In Vulkan, the depth bias state of a pipeline can be specified as dynamic, allowing it to be changed without recreating/reassigning the entire pipeline state. Given that depth bias state contains highly variable floating point fields, it would be prohibitive for performance to recreate and reassign D3D's pipeline state every time a Vulkan depth bias state is dynamically changed. To facilitate the translation and address developer requests, DDI 0099 allows the depth bias state in D3D to be specified as possibly-dynamic during pipeline state creation, as well as changing DepthBias from INT to FLOAT.

Index buffer strip cut

Vulkan's index buffer primitive restart functionality does not require or allow specifying the restart sentinel, or "strip cut value". Instead, it is always the maximum value of the index buffer's underlying type. To avoid PSO recompilation when switching index buffer views, DDI 0099 allows the buffer strip cut value to be specified as possibly-dynamic during pipeline state creation.

DDI

To legally dynamically update the depth bias state after pipeline assignment, the assigned pipeline **MUST** have been created with D3D12DDI_PIPELINE_STATE_FLAG_DYNAMIC_DEPTH_BIAS. Likewise, to legally dynamically update the IB strip cut value, the assigned pipeline **MUST** have been created with D3D12DDI_PIPELINE_STATE_FLAG_DYNAMIC_INDEX_BUFFER_STRIP_CUT.

For drivers, these flags are only *hints* about whether state is allowed to be changed; it does not alter the behavior of calls to `pfnSetPipelineState`. When setting the pipeline state, the depth bias state and IB strip cut must still be set according to their descriptions in the pipeline state object, regardless of the presence of their corresponding dynamic flags.

For pipelines created with this flag, the *current* depth bias state may be changed via the command list function, `pfnRSSetDepthBias`, and the *current* IB strip cut value may be changed via the command list function, `pfnIASetIndexBufferStripCutValue`.

Calling these functions *does not* change the pipeline state object's description. For example, if an application sets the pipeline state with depth bias state **A**, calls `RSSetDepthBias` with depth bias state **B**, and then sets the same pipeline state again, the depth bias state must then be **A**. `RSSetDepthBias` and `IASetIndexBufferStripCutValue` are both valid calls inside a bundle, and the respective state changes must be preserved upon exiting bundle execution.

DDI 0099 revs the rasterizer description to change DepthBias from INT to FLOAT. The calculation of DepthBias is not changed, as it was previously casting to a FLOAT first anyway (refer to [D3D11 spec 15.10 on Depth Bias](#)).

Support for dynamic depth bias and index buffer strip cut is implied by, and must be supported on, drivers that report their DDI version to be 0099 or greater.

Device.Graphics.AdapterRender.D3D12.DynamicPipelineState.CoreRequirement

Device.Graphics.AdapterRender.D3D12.NonNormalizedSampling

Device.Graphics.AdapterRender.D3D12.NonNormalizedSampling.CoreRequirement

D3D12 graphics hardware supporting non normalized sampling coordinates must meet the specified requirements.

Applies to**Windows 11 Client x64**

Description

Vulkan allows sampling from textures using non-normalized coordinates, as specified during sampler creation. To enable this in D3D12, DDI 0100 adds a sampler flag indicating to use non-normalized coordinates.

Samplers with D3D12_SAMPLER_FLAG_NON_NORMALIZED_COORDINATES set must sample from the target texture at the coordinates specified, unscaled by the texture's dimensions.

When non-normalized coordinates are used, the following restrictions apply:

- The SRV being sampled must be D3D12_SRV_DIMENSION_TEXTURE1D or D3D12_SRV_DIMENSION_TEXTURE2D
- The SRV's mip-count must be 1.
- The sampling coordinate valid range is x:[0, width) and y:[0, height)
- The Offset parameter in Sample* functions must not be used
- The D3D12_FILTER must be one of
 - D3D12_FILTER_MIN_MAG_MIP_POINT
 - D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT
 - D3D12_FILTER_MINIMUM_MIN_MAG_MIP_POINT
 - D3D12_FILTER_MINIMUM_MIN_MAG_LINEAR_MIP_POINT
 - D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_POINT
 - D3D12_FILTER_MAXIMUM_MIN_MAG_LINEAR_MIP_POINT

Where the min and mag filters are equal, mip is POINT, and comparison and anisotropy are disabled.

- MinLOD and MaxLOD must be 0
- AddressU and AddressV must be D3D12_TEXTURE_ADDRESS_MODE_CLAMP or D3D12_TEXTURE_ADDRESS_MODE_BORDER

The result of sampling outside of the valid range is generally undefined in terms of what value may be returned, but additionally undefined in terms of what state the device is left in if AddressU or AddressV are not appropriately set.

Support for non-normalized texture sampling is implied by, and must be supported on, drivers that report their DDI version to be 0100 or greater.

Device.Graphics.AdapterRender.D3D11TiledResource

Tiled resource area feature.

Device.Graphics.AdapterRender.D3D11TiledResource.D3D11CoreA

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

Details of this feature are in the D3D12 tiled resource spec.

Device.Graphics.AdapterRender.D3D12.TextureCopyBetweenDimensions

WDDM 3.1 drivers must support copying data between texture resources of mismatched dimension.

Device.Graphics.AdapterRender.D3D12.TextureCopyBetweenDimensions.CoreRequirement

D3D12 devices expose support for copying between texture dimensions.

Applies to	Windows 11 Client x64

Description

This feature ensures that copying is possible between dimensions of textures, e.g, a slice of a 3D texture into a whole 2D texture.

Copy between dimensions: https://dev.azure.com/cga-exchange/_git/docs?_a=preview&path=/d3d/VulkanOn12.md&version=GBmaster&fullScreen=true&anchor=texture-copy-between-inequal-dimension-textures

Device.Graphics.AdapterRender.D3D12.UnrestrictedBufferTextureCopyPitch

WDDM 3.1 drivers must support copying data between textures and buffers without requiring the data placement or pitch alignments.

Device.Graphics.AdapterRender.D3D12.UnrestrictedBufferTextureCopyPitch.CoreRequirement

D3D12 devices expose support for copying without data placement and pitch alignment requirements.

Applies to	Windows 11 Client x64

Description

This feature ensures that copying is possible between dimensions of textures, e.g, a slice of a 3D texture into a whole 2D texture.

Copy between dimensions: https://dev.azure.com/cga-exchange/_git/docs?_a=preview&path=/d3d/VulkanOn12.md&version=GBmaster&fullScreen=true&anchor=unrestricted-buffer/texture-copy-row-pitch-and-offset

Device.Graphics.AdapterRender.D3D12.UnrestrictedVertexElementAlignment

WDDM 3.1 drivers must support unaligned vertex specification during input layout creation. Vertices must still be aligned at draw time.

Device.Graphics.AdapterRender.D3D12.UnrestrictedVertexElementAlignment.CoreRequirement

D3D12 devices expose support for creating input layouts with aligned or unaligned vertex specifications.

Applies to	Windows 11 Client x64

Description

This feature relaxes validation during input layout creation that elements in the layout must be properly aligned. The elements must still be aligned once the GPU VA is resolved.

Unrestricted vertex alignment: https://dev.azure.com/cga-exchange/_git/docs?_a=preview&path=/d3d/VulkanOn12.md&version=GBmaster&fullScreen=true&anchor=unrestricted-vertex-element-alignment

Device.Graphics.AdapterRender.D3D12VideoDecoding

D3D12 video decoding feature.

Device.Graphics.AdapterRender.D3D12VideoDecoding.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

Details of this feature are described in the D3D12 protected resources spec.

Device.Graphics.AdapterRender.D3D12VideoProcessing

D3D12 video processing feature.

Device.Graphics.AdapterRender.D3D12VideoProcessing.CoreRequirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

Details of this feature are described in the D3D12 protected resources spec.

Device.Graphics.AdapterRender.D3D12VideoEncoding

D3D12 video encoding feature.

Device.Graphics.AdapterRender.D3D12VideoEncoding.CoreRequirement

Applies to	Windows 11 Client x64

Description:

Details of this feature are in the D3D12 video encoding spec.

Device.Graphics.WDDM26.AdapterRender.D3D12.DXILCore.ShaderModel64

Optional feature.

Applies to	Windows 11 Client x64

Description

Shader Model 6.4 enables new packed dot product intrinsics, support for Variable rate shading, and Library sub-objects:

- The new shader model adds the following 3 intrinsics to accelerate dot products on reduced precision data: dot4add_u8packed, dot4add_i8packed, and dot2add;
- Variable rate shading is supported as of this shader model. A new SV_ShadingRate was added that indicates the number of samples per pixel during operation of the current pixel shader thread.
- Finally, library subobject support was added as decribed in the ray tracing documentation (Direct3D 12 Raytracing - Win32 apps | Microsoft Docs)

To declare support for these features, a driver must enumerate Shader Model 6.4 or higher as the highest supported shader model. It should then accept shaders with major version 6, and minor version 4. Not all features listed are required The shader model 6.4 HLK tests must also pass.

Device.Graphics.WDDM27.AdapterRender.D3D12.DXILCore.ShaderModel65

Optional feature.

Applies to	Windows 11 Client x64

Description

Shader Model 6.5 enables DXR 1.1, sampler feedback, mesh and amplification shaders, and additional wave intrinsics.

- DirectX Raytracing (DXR) Tier 1.1 adds uint GeometryIndex(): a new intrinsic for retrieving the generated geometry index to the existing raytracing shader types with intersection information (intersection, any hit, and closest hit). A new object, RayQuery is also added to every shader stage, that enables inline access to raytracing operations.
- Sampler Feedback is an optional feature available in shader model 6.5 that adds 2 new Texture resource types: FeedbackTexture2D<type> and FeedbackTexture2DArray<type> to HLSL. These resource types have a template argument specifying the format of the feedback map. See the HLSL section of the Sampler Feedback spec for details.
- Two new shader types are added to HLSL for the new mesh shader graphics pipeline. These are Mesh Shaders ms_6_5 and Amplification Shaders as_6_5. See the Mesh Shader spec for details.
- WaveMatch(), and a number of WaveMultiPrefix*() intrinsics, are available starting with shader model 6.5, when the optional Wave Intrinsics feature is supported. All shader stages, except for Raytracing shaders, support wave intrinsics as of shader model 6.5.
- Finally, library subobject support was added as described in the ray tracing documentation (Direct3D 12 Raytracing - Win32 apps | Microsoft Docs)

To declare support for these features, a driver must enumerate Shader Model 6.5 or higher as the highest supported shader model. It should then accept shaders with major version 6, and minor version 5. The shader model 6.5 HLK tests must also pass.

Device.Graphics.IndirectDisplay.Wired

Display feature requirements for wired indirect displays.

Device.Graphics.IndirectDisplay.Wired.Base

Requirements for a wired indirect display.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

An indirect display is a display device (e.g. monitor) connected to a remote Indirect Display device that receives the image from a user mode driver conforming to Microsoft's Indirect Display framework on the sender system, performs some processing and then displays the image on the monitor.

On sender system the display image needs CPU interaction in order to be processed and transmitted to the Indirect Display device. This CPU interaction may be direct manipulation of the image or just programming hardware to process the image.

Some examples are, but not limited to, universal docking stations, USB display dongles and Monitors with USB display capabilities.

Common Indirect Display DDIs

Indirect displays must implement all required DDI's listed for indirect display drivers in the WDK.

Requirements:

UMDF and DevFund Requirements: An indirect display driver must be a UMDF driver and must adhere to all pertinent Device Fundamental requirements. The following DevFund Driver Framework UMDF requirements must be met, but not limited to.

- Device.DevFund.DriverFramework.UMDF
 - Device.DevFund.DriverFramework.UMDF.Reliability
 - Device.DevFund.DriverFramework.UMDF.WDFProperINF

Graphics Requirement: Indirect display drivers must adhere to the follow graphics requirements.

- Device.Graphics.AdapterBase.PowerManagementCompliance - This should come from the USB Req
- Device.Graphics.AdapterBase.Symbols
- Device.Graphics.WDDM.Display.GammaCorrection – (optional requirement)
- Device.Graphics.WDDM.Display.I2CSupport – (optional requirement)
- Device.Graphics.WDDM.Display.Multimon
- Device.Graphics.WDDM12.Display.ContainerIDSupport
- Device.Graphics.WDDM12.Display.ModeEnumeration
- Device.Graphics.WDDM.DisplayRender.OutputProtection

HDCP support is optional but the device must implement OPM and have an appropriate OPM certificate.

Device.Graphics.MCDM27

MCDM 2.7 is a required version level for MCDM drivers. If a driver implements MCDM, then it must implement all of the MCDM 2.7 Requirements except those explicitly marked as optional.

Device.Graphics.MCDM27.CoreRequirement

Compute Accelerator devices must support the MCDM model, features, and workload.

Applies to	Windows 11 Client x64

Description

Details of this requirement are in the MCDM spec.

Device.Graphics.MCDM27.OneCore

MCDM 2.7 drivers must support OneCore.

Device.Graphics.MCDM27.OneCore.CoreRequirement

MCDM 2.7 drivers must be able to load and run compute workloads on OneCore-based devices, both natively and under paravirtualization.

Applies to	Windows 11 Client x64

Description

Details of this requirement are available in a MCDM specification. The requirement is outlined below:

- The UMDs must be compiled against the OneCore libs and function correctly on OneCore-based OSes.
- The KMD must be compiled against the OneCore libs and function correctly on OneCore-based OSes.
- The driver must support paravirtualization on OneCore-based OSes, as the host and as the guest, in compatibility mode.

Device.Graphics.MCDM27.Containers

MCDM 2.7 drivers must support running inside of Windows Server Containers as well as Hyper-V Containers.

Device.Graphics.MCDM27.Containers.CoreRequirement

MCDM 2.7 drivers must be able to load and run compute workloads within Windows containers.

Applies to	Windows 11 Client x64

Description

Details of this requirement are available in the MCDM specification.

Drivers must function correctly while running inside of a container environment where they are isolated from the host file system and registry, and must ensure that they use the appropriate kernel services to access any necessary data.

Device.Graphics.WDDM

The base feature set implemented by drivers supporting all versions of the WDDM.

Device.Graphics.WDDM.Base

Graphics drivers must be implemented per the WDDM 1.0 spec.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client 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

A 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 11 Client 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

Device.Graphics.WDDM.Display

The base feature set implemented by drivers supporting all versions of the WDDM Display DDIs.

Device.Graphics.WDDM.Display.Base

Graphics drivers must be implemented per the WDDM 1.0 specification.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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. This ability must be supported for RGBA8888 and RGBA1010102 pixel formats.

Device.Graphics.WDDM.Display.HardwareCaptureValidation

Graphics drivers must pass a test suite using a hardware display loopback device.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Display devices implementing the WDDM specification must pass a series of tests using a hardware loopback device approved by Microsoft. These tests are aimed at validating the implementation of hardware color pipeline, multi-plane overlays, dynamic mode management, EDID and DisplayID parsing, and more.

Device.Graphics.WDDM.Display.HotPlugDetection

A graphics adapter must reliably detect the connect and disconnect event of display devices.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](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](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](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 11 Client 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](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 11 Client 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 itself, with respect to resolution, Direct 3D, 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.

Device.Graphics.WDDM.DisplayRender

The base feature set implemented by drivers supporting all versions of the WDDM for both Display and Render DDIs.

Device.Graphics.WDDM.DisplayRender.Base

Graphics drivers must be implemented per the WDDM 1.0 specification.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64

Windows 11 Client ARM64

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.

Device.Graphics.WDDM.Render

The base feature set implemented by drivers supporting all versions of the WDDM Render DDIs.

Device.Graphics.WDDM.Render.Base

Graphics drivers must be implemented per the WDDM 1.0 specification.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

See requirement **Device.Graphics.WDDM.Base**

Device.Graphics.WDDM.Render.VideoProcessing

A display WDDM driver must support the DirectX VA 2.0 Video Processor DDI.

Applies to	Windows 11 Client x64

Terms: If-Implemented

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>.

Device.Graphics.WDDM11

The base feature set implemented by drivers supporting WDDM 1.1.

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.WDDM11.Display

The base feature set implemented by drivers supporting all versions of the WDDM Display DDIs.

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 11 Client x64 Windows 11 Client ARM64

Description

See requirement **Device.Graphics.WDDM11.Base**

Device.Graphics.WDDM11.DisplayRender

The base feature set implemented by drivers supporting all versions of the WDDM for both Display and Render DDIs.

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 11 Client x64 Windows 11 Client ARM64

Description

See requirement **Device.Graphics.WDDM11.Base**

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.

Device.Graphics.WDDM11.DisplayRender.D3D9Overlay.D3D9Overlay

A WDDM1.1 driver must support Direct3D 9 overlays.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.WDDM11.Render

The base feature set implemented by drivers supporting all versions of the WDDM Render DDIs.

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 11 Client x64

Description

See requirement **Device.Graphics.WDDM11.Base**

Device.Graphics.WDDM11.Render.DXVAHD

The optional feature that is implemented by WDDM 1.1 drivers supporting the new state-based video processing DDIs.

Device.Graphics.WDDM11.Render.DXVAHD.DXVAHD

WDDM1.1 driver supports DXVA-HD

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

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

Device.Graphics.WDDM12

The base feature set implemented by drivers supporting WDDM 1.2.

Device.Graphics.WDDM12.Base

Graphics drivers must be implemented per the WDDM 1.2 specification.

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

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 Graphic s	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).

Device.Graphics.WDDM12.Display

Display feature requirements for all WDDM12 drivers that support the display specific DDIs

Device.Graphics.WDDM12.Display.Base

Requirements for a WDDM graphics adapter to support display functionality

Applies to	Windows 11 Client 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 **
DxgkDdiSupportedVidPn
DxgkDdiRecommendFunctionalVidPn ***
DxgkDdiEnumVidPnCofuncModality
DxgkDdiSetVidPnSourceVisibility
DxgkDdiCommitVidPn
DxgkDdiUpdateActiveVidPnPresentPath
DxgkDdiRecommendMonitorModes
DxgkDdiGetScanLine
DxgkDdiQueryVidPnHWCapability
DxgkDdiPresentDisplayOnly
DxgkDdiReleasePostDisplayOwnership
DxgkDdiSystemDisplayEnable
DxgkDdiSystemDisplayWrite
DxgkDdi2CReceiveDataFromDisplay
DxgkDdi2CTransmitDataFromDisplay

*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 11 Client x64 Windows 11 Client ARM64

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 11 Client 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](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

Page **210** of **561**

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 (PositionX, 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 11 Client x64 Windows 11 Client ARM64

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 driver 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 11 Client x64 Windows 11 Client ARM64

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 (`D3DDDI_FMT_R8G8B8`) or 32bpp (`D3DDDI_FMT_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 11 Client 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:

1. Ensure updates to the frame buffer must be scanned out without requiring addition IHV/OEM.
2. software/drivers. Provide a linear frame buffer that is CPU accessible on demand from the IHV driver.
3. On UEFI systems at boot using UEFI 2.3 GOP.
4. On BIOS systems using VBE 3.0 standards.

Device.Graphics.WDDM12.DisplayOnly

The optional feature set implemented by WDDM 1.2 drivers that support only the display specific DDIs.

Device.Graphics.WDDM12.DisplayOnly.Base

Requirements for a WDDM graphics adapter to support display functionality

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](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 **
DxgkDdiSupportedVidPn
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 D3DDDI_FMT_A8R8G8B8; therefore, these drivers should only enumerate source modes of this format.

Device.Graphics.WDDM12.DisplayRender

The optional feature set implemented by WDDM 1.2 drivers supporting both display and render DDIs.

Device.Graphics.WDDM12.DisplayRender.Base

Requirements for a WDDM graphics adapter implementing both Render and Display DDIs

Applies to	Windows 11 Client 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](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 **
DxgkDdiSupportedVidPn
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

Device.Graphics.WDDM12.DisplayRender.ProcessingStereoscopicVideoContent

The optional feature set implemented by WDDM 1.2 drivers that support stereoscopic video processing.

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.WDDM12.DisplayRender.RuntimePowerMgmt

The optional feature set implemented by WDDM 1.2 drivers that support the runtime power management DDIs.

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.WDDM12.Render

The base feature set implemented by WDDM 1.2 drivers that support the render specific DDIs.

Device.Graphics.WDDM12.Render.Base

Requirements for a WDDM graphics adapter to support render functionality

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](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 11 Client x64 Windows 11 Client ARM64

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

If-implemented: 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.

If-implemented: VPx modes:

- D3D11_DECODER_PROFILE_VP8_VLD
- D3D11_DECODER_PROFILE_VP9_VLD_PROFILE0

- D3D11_DECODER_PROFILE_VP9_VLD_PROFILE2 (optional if hardware does not support profile 2)

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 MPEG2 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 11 Client x64 Windows 11 Client ARM64

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_CONSTRICION
 - D3D11_1DDI_VIDEO_PROCESSOR_FEATURE_CAPS_CONSTRICION
- 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 11 Client x64 Windows 11 Client ARM64

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_SETVIDVIDPNOURCEADDRESS 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.FlipOnVSyncMmlo

WDDM1.2 drivers must support a memory mapped I/O (MMIO)-based flip that takes effect on the next vertical sync.

Applies to	Windows 11 Client x64

Description

All WDDM1.2 drivers must publish the FlipOnVSyncMmlo field as TRUE in the DXGK_FLIPCAPS structure, and implement behavior outlined under FlipOnVSyncMmlo here: [http://msdn.microsoft.com/en-us/library/ff561069\(v=VS.85\).aspx](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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client 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:

1. Catalog signature with an overall attribute of PE TRUSTED, with a hash entry for each specific file setting a field and value of PETrust=1.
2. 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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64

Description

To ensure a quality printing experience on Windows, the WDDM1.2 graphic drivers must support XPS Rasterization.

Device.Graphics.WDDM12.RenderOnly

The optional feature set implemented by WDDM 1.2 drivers which support only the render specific DDIs.

Device.Graphics.WDDM12.RenderOnly.Base

Requirements for a WDDM graphics adapter to support render functionality

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](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.StandbyHibernateFlags

The optional feature implemented by WDDM 1.2 drivers supporting the new stand by hibernation flags.

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Graphics.WDDM13

The base feature set implemented by drivers supporting WDDM1.3.

Device.Graphics.WDDM13.Base

Graphic drivers must implement WDDM1.3.

Applies to	Windows 11 Client 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	NA	Mandatory

		Optional for DX10+		
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

Device.Graphics.WDDM13.DisplayRender

The parent feature node of requirements that have an impact for the display and render GPU.

Device.Graphics.WDDM13.DisplayRender.48HzVideoPlayback

WDDM1.3 drivers must support 48 Hz Video Playback.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Graphic Drivers implementing WDDM1.3 must implement DDIs that:

1. Report whether the collective system can seamlessly switch to 48 Hz (intersection of SOC and display panel capabilities).
2. If so, allow the refresh rate (such as 48 Hz) to be specified on each particular Present DDI.

The HCK 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 that report support for 48 Hz or 50 Hz through the capabilities DDI through an HCK test, which is described in detail in the 48 Hz IHV spec.

Device.Graphics.WDDM13.DisplayRender.IndependentFlip

IndependentFlip support

Applies to	Windows 11 Client 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.

Device.Graphics.WDDM13.DisplayRender.MultiplaneOverlaySupport

Multi-plane Overlay support

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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:

1. 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.
- 2. 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.
- 3. 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.
- 4. The DWM's swapchain (plane 0) must be able to interact with the other overlay planes.
- 5. All planes must be able to be enabled and disabled, including plane 0 (the DWM's swapchain).
- 6. All planes must support source and destination clipping, including plane 0 (the DWM's swapchain).
- 7. At least one plane must support shrinking and stretching, independent from other planes that may be enabled.
- 8. Planes that support scaling must support both bilinear filtering and better than bilinear filtering quality.
- 9. 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).
- 10. 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.
- 11. 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;

```

- o 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.DisplayRender.CoolingInterface

Feature for Thermal Hints

Device.Graphics.WDDM13.DisplayRender.CoolingInterface.ThermalHints

Optional Thermal Hints support for WDDM1.3 drivers

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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, WHCK can receive the error log that the miniport driver fails the DxgkDdiQueryInterface call on GUID_THERMAL_COOLING_INTERFACE and fail the WHCK test.

Device.Graphics.WDDM13.DisplayRender.WirelessDisplay

The optional feature set implemented by WDDM 1.3 drivers that support the Miracast functionality.

Device.Graphics.WDDM13.DisplayRender.WirelessDisplay.BasicWirelessDisplay

Optional Wireless Display support for WDDM1.3 drivers

Applies to**Windows 11 Client x64**

Description

NOTE: In Windows 10 the Wireless Display framework was changed significantly so that most of the work is done by the OS. The requirements in this section do not apply to WDDM 2.0 Drivers. However, existing WDDM 1.3 Drivers can continue to use this mechanism.

- 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 WHCK.
- 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.

Device.Graphics.WDDM13.EnhancedPowerManagement

Graphics Kernel Power Management Enhancements

Device.Graphics.WDDM13.EnhancedPowerManagement.FState

F-State Hysteresis

Applies to	Windows 11 Client 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 11 Client x64

Description

WDDM 1.3 introduces some power management enhancements:

IN PHASE VSYNC

While the Vsync is enabled, 98% of the Vsycns must be within 500 us of the expected value with no Vsycns exceeding a variance of more than 1.5 ms.

Device.Graphics.WDDM13.Render

The base feature set implemented by WDDM 1.3 drivers that support the render specific DDIs.

Device.Graphics.WDDM13.Render.CheckDDIBoundaries

Hybrid Graphics

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.GPUNode

Graphics Kernel Performance

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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.TrimSupport

Direct3D Trim

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

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.

Device.Graphics.WDDM20

Device.Graphics.WDDM20.CoreRequirement

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Terms: If-Implemented

Description

1. 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

2. 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	

3. 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.

Device.Graphics.WDDM20.Core

The base feature set implemented by drivers supporting WDDM2.0.

Device.Graphics.WDDM20.Core.Base

WDDM2.0

Applies to

Windows 11 Client x64

Terms: If-Implemented

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

Device.Graphics.WDDM20.DisplayRender

The parent feature node of requirements that have an impact for the display and render GPU.

Device.Graphics.WDDM20.DisplayRender.HardwareContentProtection

A GPU must support hardware-based content protection.

Applies to	Windows 11 Client 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.

Device.Graphics.WDDM20.Display.VirtualModeSupport

Device.Graphics.WDDM20.Display.VirtualModeSupport.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

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.

Device.Graphics.WDDM21

WDDM 2.1 is an optional version level introduced for Windows 10 builds greater than 10586. If a driver implements WDDM 2.1 then it must implement all of the WDDM 2.1 Requirements. WDDM 2.0 drivers will continue to run on the newer Windows 10 builds.

Device.Graphics.WDDM21.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

A WDDM 2.1 driver must meet all requirements defined in the WDDM 2.1 Specification. In particular, drivers should implement the following (depending on hardware capabilities):

1. Offer / Reclaim

WDDM 2.1 drivers must support the changes to the Offer / Reclaim feature.

2. Video Protected Region

Hardware that implements a single-range VPR (Video Protected Region) must support the new Move Paging DDIs so that the VPR can be defragmented.

3. Large Page PTE Encoding

If the hardware supports large page PTEs, e.g. 64KB, these are supported by WDDM 2.1 and should be implemented.

4. Pre/Post Present Optimizations

A new callback in WDDM 2.1 (UpdateAllocationProperties) allows for scenarios involving Pre/Post Present optimizations. This is optional.

Device.Graphics.WDDM21.FlipQueueLatency

Applies to

Windows 11 Client x64

Terms: If-Implemented

Description:

A WDDM2.1 driver must support a 64bit PresentID member in DXGK_MULTIPANE_OVERLAY_VSYNC_INFO2 assigned to each flip request for a particular MPO plane. This value will be passed to the driver at the time of the flip request, and the driver must report back the currently acknowledged PresentId value in the VSync ISR callback.

The ability to queue more than one flip-on-VSync request on the same MPO plane must be controlled by the MaxQueuedMultiPlaneOverlayFlipVSync member in DXGK_DRIVERCAPS.

For hardware that cannot support more than one in flight flip-on-VSync request, the driver must set MaxQueuedMultiPlaneOverlayFlipVSync to 1.

If the OS passes a non-negative MaxImmediateFlipLine (member of DXGK_MULTIPANE_OVERLAY_PLANE3) to the driver and the driver detects that the hardware has not yet started to scan out of horizontal line $\geq$ MaxImmediateFlipLine, the driver must perform an immediate flip for the new surface.

In the case where a converted immediate flip overrides a previous pending flip-on-VSync, the driver must report the immediate flip conversion success.

The driver must pass GPU clock and frequency values in the VSync ISR callback (DXGKARGCB_NOTIFY_INTERRUPT_DATA).

Device.Graphics.WDDM22

WDDM 2.2 is an optional version level introduced for Windows 10 version 1607 builds greater than 14393. If a driver implements WDDM 2.2 then it must implement all of the WDDM 2.1 Requirements. WDDM 2.0 drivers will continue to run on the newer Windows 10 builds

Device.Graphics.WDDM22.DisplayRender

If implementing WDDM 2.2, full graphics drivers must implement the following features.

Device.Graphics.WDDM22.DisplayRender.4HWID

Applies to

Windows 11 Client x64

Business Justification:

WDDM 2.2 enables a broad feature set across new hardware features such as HDR, Windows Holographic and several other software improvements. To enable these premium experiences on a variety of Windows PC form factors, WDDM 2.2 graphics drivers from graphics IHVs must include 4-part Hardware ID targeting in a single driver package.

Description:

The WDDM 2.2 graphics driver installation file (INF) must contain 4-part Hardware IDs. An example of such targeting in the graphics driver INF is "PCI\VEN_1000&DEV_0001&SUBSYS_00000001". This unified graphics driver package must include system configuration settings which are limited to the INF file provided by the graphics IHV. Microsoft & the graphics IHVs will independently service these components for ongoing support of these features.

WDDM 2.2 graphics drivers must not use CHIDs for targeting through Windows Update. CHID usage is acceptable only if there is a strict dependency of a particular graphics driver on the firmware or other drivers which are not included in the graphics driver package. Any other usage of CHIDs will result in the revocation of the system certification. System manufacturers are encouraged to resolve the dependencies between the graphics driver and other system component drivers with their graphics IHV so as to enable a consistent and smooth upgrade experience for subsequent Windows releases.

Device.Graphics.WDDM22.DisplayRender.PeriodicFrameNotification

Applies to

Windows 11 Client x64

Description:

Periodic frame notifications are used in VR scenarios to allow the VR compositor Last Stage Re-projection (LSR) GPU work be scheduled within a fixed time ahead of every VSYNC in order to minimize overall latency between head motion and photon hitting the user's eye. This is a major improvement over what is possible to do today using VSYNC synchronization, shaving in the order of 10ms worth of latency.

Driver are required to implement the new kernel mode DDI introduced by this feature, typically leveraging programmable VLine interrupt capabilities on the hardware.

Business Justification: VR scenarios are becoming increasingly common both from third party as well as first party with the upcoming Windows Holographics. Periodic frame notification is a critical part of reducing overall perceived latency and for providing end user a great VR experience.

Device.Graphics.WDDM22.DisplayRender.DisplayWireFormatInfo

Applies to

Windows 11 Client x64

Description:

In previous DDIs, wire format was left as a detail for the driver implementer to decide based on the source pixel format and the display output capabilities. This ambiguity meant that the OS had no way to know or control if a mode change would be a hard mode change requiring a reconfiguration of the pixel stream with the display device and a likely multi-frame loss of signal, or a soft mode change involving only display controller changes and/or parameter changes on the display device.

The driver is required to report the specific timings for each monitor mode as before, however now the supported bit depth and color encoding format sent to the monitor must be included.

Specifically, the driver must report the specific timings for each monitor mode through the updated DxgkDdiEnumVidPnCofuncModality function.

Business Justification:

This requirement is one of many required to support High Dynamic Range (HDR).

IHVs and OEMs are looking for ways to differentiate in order to garner higher margins beyond commodity devices and to delight their customers. In gaming, video, and other areas the latest technology advancement is UHD (Ultra-high definition). Although 4K is currently the best known manifestation of this, HDR (along with wider color gamuts) provides the real payoff with this technology, allowing for “ultra-realistic” games, videos, and creation scenarios.

Device.Graphics.WDDM22.DisplayRender.MultiPlaneOverlay3

Applies to**Windows 11 Client x64**

Description:

Extending the current MultiPlaneOverlay (MPO) implementation to simplify the presentation path as well as support a variety of new features in Windows 10, version 1703.

In some scenarios, Windows will use multi-plane overlays to emulate mode changes without performing a real mode change. Since not all hardware supports multi-plane overlay capabilities, the multi-plane overlay DDI is extended to include support for hardware scalers that stretch the final image to fit the monitor (sometimes referred to as panel fitters).

Additionally, HDR will be supported using some of the existing MPO DDIs and by extending some others.

Specifically, drivers must:

- implement `pfnPresentMultiPlaneOverlay1` DDI
- implement `DXGKDDI_SETVIDPNSOURCEADDRESSWITHMULTIPLANEOVERLAY3` DDI
- implement `DXGKDDI_CHECKMULTIPLANEOVERLAYSUPPORT3` DDI

Business Justification:

This requirement is also one of many required to support High Dynamic Range (HDR) and Panel Fitters.

IHVs and OEMs are looking for ways to differentiate in order to garner higher margins beyond commodity devices and to delight their customers. In gaming, video, and other areas the latest technology advancement is UHD (Ultra-high definition). Although 4K is currently the best known manifestation of this, HDR (along with wider color gamuts) provides the real payoff with this technology, allowing for “ultra-realistic” games, videos, and creation scenarios.

Device.Graphics.WDDM22.DisplayRender.SetTimings

Applies to**Windows 11 Client x64**

Description:

The new DDI to control setting up the display configuration is `SetTimingsFromVidPN` which replaces the `CommitVidPN` call used in previous WDDM versions. The new DDI differs from the old one in multiple respects but specifically relating to the MPO DDI convergence, the allocation handle for the base plane is not passed to the new DDI and there is no longer a requirement that a frame buffer be allocated for the base plane prior to enabling or modifying a path.

SetTimingsFromVidPn always describes the final state intended by the OS which allows the driver to optimize how hardware is assigned and also leaves the driver in control of how to perform a combination of changes rather than working through an intermediate state from the OS.

Specifically, the driver must implement the DXGKARG_SETTIMINGSFROMVIDPN DDI.

Business Justification:

This requirement is one of many required to support High Dynamic Range (HDR).

IHVs and OEMs are looking for ways to differentiate in order to garner higher margins beyond commodity devices and to delight their customers. In gaming, video, and other areas the latest technology advancement is UHD (Ultra-high definition). Although 4K is currently the best known manifestation of this, HDR (along with wider color gamuts) provides the real payoff with this technology, allowing for “ultra-realistic” games, videos, and creation scenarios.

Device.Graphics.WDDM22.DisplayRender.DisplayDetection

Applies to**Windows 11 Client x64**

Description:

Connection state issues can cause unexpected device behavior for users, such as windows being rearranged or resized. The DisplayDetection feature requests a detailed report of connection state changes from the driver to the OS to better react to the myriad of ways that the connection state can change.

By exposing the specifics of the activity to the OS, there is less pressure on the driver to account for all the possible changes and let the OS react more intelligently. This is more prevalent for DisplayPort Link Training since the expectation is that the connection will only disappear for a short time to complete the capability query. Having knowledge that link training is occurring allows the OS to expect a short loss of connection that is expected to come back, which means it can avoid changing to the mode with that display disconnected.

Also, issues can arise when multiple displays are connected simultaneously and so we'd like to highlight some changes to help avoid those. Namely, the driver must report any targets inherent to the adapter and new targets are reported when connected to targets that were previously reported. It is also important that when multiple displays are reported to be connected simultaneously, the displays should be reported all together as a batch instead of sending them one by one.

Finally, there can be cost savings by front-loading any integrated displays to the PnP. Currently, they are exposed by the driver in the same way that external displays are, even though their properties won't change. During driver initialization, the OS will query for the descriptors, native display timing, monitor link info, and other static attributes.

Business Justification:

This issue can create a lot of customer dissatisfaction due to the annoyance it can cause since users must manually move and resize their windows to recreate their original setup.

Furthermore, it can also create the illusion that these issues are caused by their monitor or PC/OS and attempt to return these when the issue could actually just be caused by a faulty connection.

Device.Graphics.WDDM22.DisplayRender.ReducedBlackScreen

Applies to**Windows 11 Client x64**

Description:

This feature makes changes in the boot sequence so that users can get a smoother and less glitchy boot experience. Currently, there are two periods of time when a user starts up their machine when they are subjected to a completely blank screen. The first period of blackscreen, which occurs during the transition from UEFI to graphics driver initialization, can be eliminated with collaboration between the driver and OS to ensure content is always being shown and the transition itself is seamless. Furthermore, this feature calls for changes that help bridge the continuity during this transition by inheriting the hardware display settings.

One of the reasons this first blackscreen exists is because there is a period of time when the UEFI frame buffer stops driving the display before the OS is able to render content when the driver is initializing. This feature calls for the UEFI frame buffer to remain available and to continue driving the display during driver initialization so content will always be available. Furthermore, to eliminate a glitchy transition, the graphics driver should switch away from using the UEFI frame buffer on the first present of the OS provided frame buffer.

The other aspect of ensuring a seamless transition is that the two frame buffers are outputting content with the same display settings (i.e. if the UEFI frame buffer is rendering at 480p and the OS provided frame buffer is at native 1080p, then the jump in resolution won't look smooth to the user). To accomplish this, the graphics driver and UEFI GOP component have to coordinate such that the driver is able to inherit the display mode set by the UEFI GOP (as well as its contents for render continuity). Additionally, the OS must be able to know when the driver is capable of inheriting these display settings and be able to query for these settings as well.

Business Justification: A glitchy and inconsistent boot sequence hurts the quality feel of premium devices by making them feel less refined than their competition. It is a fairly ugly experience when users see no content for extended periods of time, changing resolutions, different color schemes, etc. This work to improve the feeling of booting up a device helps improve user experiences with their OS and machines.

These periods of blackscreen have also led to user concerns that their machines may be crashing when they aren't seeing anything being rendered to their displays. By constantly drawing content to the user, they are given more evidence that the system is still operating properly.

Device.Graphics.WDDM22.DisplayRender.HighDynamicRange

Device.Graphics.WDDM22.DisplayRender.HighDynamicRange.CoreRequirement

Applies to

Windows 11 Client x64

Terms: If-Implemented

Note: High Dynamic Range is an optional WDDM feature. There are several HLK requirements under the HighDynamicRange category, that are listed as "Required if HDR is implemented". It's important to note that none of these requirements apply to drivers that do not implement HDR. Any WDDM driver that *does* implement HDR, must implement the HDR required subset of features.

Description:

High Dynamic Range (HDR) support requires the following changes:

- A DDI to report the proper monitor information, including whether HDR is supported and the maximum nits supported by the display.
- A mechanism for the application to know whether a specific colorspace is supported.

- A mechanism to pass the colorspace information of the plane.
- A mechanism to pass the HDR metadata describing how the content was authored to the display output.
- A mechanism that allows the HW to switch between fp16 to 10:10:10:2 and back again without causing any major artifacts (e.g. full mode changes that blank the monitor).

Specifically, the driver must:

- report the specific timings for each monitor mode through the updated DxgkDdiEnumVidPnCofuncModality function
- implement the DXGKARG_SETTIMINGSFROMVIDPN DDI
- implement pfPresentMultiPlaneOverlay1 DDI
- implement DXGKDDI_SETVIDPNSOURCEADDRESSWITHMULTIPLANEOVERLAY3 DDI
- implement DXGKDDI_CHECKMULTIPLANEOVERLAYSUPPORT3 DDI
- support HDR metadata and colorsapce information passed to these KM DDIs
- support and indicate flipping between FP16, SDR 8bit RGB, and HDR10 allocations

In addition, for video playback:

- VPBlt must have the following format color space conversion support:

Format In DXGI_FORMAT_	Color Space In DXGI_COLOR_SPACE_	Format Out DXGI_FORMAT_	Color Space Out DXGI_COLOR_SPACE_	Required For
P010	YCBCR_STUDIO_G2084_LEFT_P2020 YCBCR_STUDIO_G22_LEFT_P2020 YCBCR_STUDIO_G22_LEFT_P709	R16G16B16A16_FLOAT	RGB_FULL_G10_NONE_P709	HDR with FP16 DWM composed desktop.
P010	YCBCR_STUDIO_G2084_LEFT_P2020 YCBCR_STUDIO_G22_LEFT_P2020 YCBCR_STUDIO_G22_LEFT_P709	B8G8R8A8_UNORM	RGB_FULL_G22_NONE_P709	HDR with sRGB DWM composed desktop.

- The new DDI VideoProcessorSetStreamHDRMetaData for metadata type DXGI_HDR_METADATA_TYPE_HDR10 must be supported
- DXVA profile D3D11_DECODER_PROFILE_HEVC_VLD_MAIN10 must be supported.
- It's also recommended that D3D11_DECODER_PROFILE_VP9_VLD_10BIT_PROFILE2 is supported.

Business Justification:

IHVs and OEMs are looking for ways to differentiate in order to garner higher margins beyond commodity devices and to delight their customers. In gaming, video, and other areas the latest technology advancement is UHD (Ultra-

high definition). Although 4K is currently the best known manifestation of this, HDR (along with wider color gamuts) provides the real payoff with this technology, allowing for “ultra-realistic” games, videos, and creation scenarios.

Device.Graphics.WDDM22.DisplayRender.ColorManagement

Device.Graphics.WDDM22.DisplayRender.ColorManagement.CoreRequirement

Applies to**Windows 11 Client x64**

Terms: If-Implemented

Description:

The display colorspace transform feature allows an OEM or third party to perform more sophisticated display color calibration and color enhancement than was previously possible using legacy gamma ramps. This feature exposes a pipeline with 1DLUTs and color matrix that operates on the final/flattened framebuffer before it is scanned out to the display.

In Windows 10, version 1703 the CSC transform is exposed to applications in a very restricted capability, specifically only for display calibration based on ICC profiles. Other scenarios traditionally enabled by legacy gamma ramps, such as fullscreen effects like fade in/out, are out of scope for Windows 10, version 1703.

Business Justification:

Display color accuracy is transitioning from a key differentiator for premium PCs to being mandatory for a successful premium device. OEMs and third party display calibration vendors have used a combination of industry standard gamma LUTs, which are not expressive enough to perform all the color transforms needed, or IHV-specific solutions, which are only available on specific hardware platforms.

This feature exposes a richer set of colorspace transform capabilities in a hardware-agnostic way that is fully integrated into the main OS. It also opens up non desktop form factors.

Device.Graphics.WDDM22.Render.BrowserSecurity

Device.Graphics.WDDM22.Render.BrowserSecurity.ArbitraryCodeGuard

User mode drivers must be built in a way that supports Arbitrary Code Guard.

Applies to**Windows 11 Client x64**

Description:

Arbitrary Code Guard (ACG) is a security mitigation that, when applied to a process, prevents that process from allocating any new code pages.

In order to be ACG compliant, the user mode driver needs to be compiled such that the driver does not emit pages that are both writeable and executable or are writable and then later become executable. To ensure proper

compilation, the user mode driver should not merge the import table into an executable code section. This can be prevented by making sure that the /merge flag of the Visual C++ linker is not set to merge the ".rdata" section into a code section. The compiled binary should also have a SectionAlignment value that is at least equal to its PAGE_SIZE.

In addition to proper compilation, user mode drivers should also either entirely remove JIT code or use out-of-proc JIT code generation.

Device.Graphics.WDDM22.Render.BrowserSecurity.ControlFlowGuard

User mode drivers must be built in a way that supports Control Flow Guard.

Applies to	Windows 11 Client x64
Description:	
The /guard:cf option causes the compiler and linker to analyze control flow for indirect call targets at compile time, and then to insert code to verify the targets at runtime. By default, /guard:cf is off and must be explicitly enabled.	
If the check fails at runtime on a CFG-aware operating system, the operating system closes the program.	
<u>Design Notes:</u>	
CFG-enabled code works fine on "CFG-unaware" versions of Windows and is therefore fully compatible with them.	
Business Justification:	

Although all Windows 10 in-box binaries are protected by Control Flow Guard (CFG) today, many non-Microsoft drivers which lack CFG protection continue to get loaded in the context of user mode processes. To close the gap in memory corruption mitigation coverage created by such 3rd party drivers not having CFG enabled, our first step with this requirement is to promote CFG's adoption across all graphics drivers loaded into browser processes, which were targeted due to the browser's prominence as a Windows attack vector

Device.Graphics.WDDM22.AdapterRender.D3D11.ParametrizedSwizzlesR1

Device.Graphics.WDDM22.AdapterRender.D3D11.ParametrizedSwizzlesR1.CoreRequirements

Applies to	Windows 11 Client x64
Description:	
If implementing D3D11 for WDDM 2.2, graphics drivers must divulge the address swizzle patterns used for textures by the GPU.	
Business Justification:	
Tools need simpler ways to capture & playback the memory used by GPUs. Microsoft middleware needs UMA optimizations, but doesn't require standard swizzle. Games are interested in uploading resource data with the CPU and without GPU involvement.	

Device.Graphics.WDDM22.AdapterRender.D3D12Core

D3D12 is an optional feature for WDDM 2.2 drivers, however, if implementing D3D12 in WDDM 2.2, drivers must support these features:

Device.Graphics.WDDM22.AdapterRender.D3D12Core.HeapPrioritiesR1

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

Drivers are required to set priorities such that application gain some amount of control over the effective heap priority. This gives applications some amount of control over demotion and promotion without having to implement it manually.

Business Justification:

D3D12 applications need simpler ways to manage heap residency, in order to quickly support GPUs with diverse amounts of physical memory. Applications may gain enough control to minimize variability when stretching 2GB scenarios to run with UltraHD resolutions or higher MSAA settings. Or, applications may even be able to lower their physical memory minimum adapter requirements.

Device.Graphics.WDDM22.AdapterRender.D3D12Core.ParametrizedSwizzlesR1

Applies to	Windows 11 Client x64

Description:

Graphics drivers must divulge the address swizzle patterns used for textures by the GPU.

Business Justification:

Tools need simpler ways to capture & playback the memory used by GPUs. Microsoft middleware needs UMA optimizations, but doesn't require standard swizzle. Games are interested in uploading resource data with the CPU and without GPU involvement.

Device.Graphics.WDDM22.AdapterRender.D3D12Core.LateLatch

Applies to	Windows 11 Client x64

Description:

Late-Latching requires atomic GPU memory operations as well as some control over GPU engine command dispatching & GPU cache invalidation.

Business Justification: These changes improve efficiency and reduce latency for Windows Holographic scenarios.

Device.Graphics.WDDM22.AdapterRender.D3D12Core.RealtimeCommandQueues

Applies to	Windows 11 Client x64

Description:

The GPU must begin execution of a command submitted on a 'realtime' queue within 5ms of the ID3D12CommandQueue::ExecuteCommandList being invoked on the CPU. Ideally, the GPU will begin execution much more rapidly than that, but this is believed to be a sufficient 'padding' value to overcome test noise.

If an adapter does not report support for `>= DXGI_GRAPHICS_PREEMPTION_PRIMITIVE_BOUNDARY` AND `DXGI_COMPUTE_PREEMPTION_DISPATCH_BOUNDARY` preemption, AND the hardware supports a limited form of preemption that can be exposed to applications, the UMD must, in response to a flag from the runtime, allow a D3D12 Command Queue to be created that 'globally' preempts all other work on the system.

- Note: No action is needed from UMD if it is already reporting support for at least both of the preemption levels above, the D3D12 runtime will leverage that existing support.

Additionally, the whole point of preemption/real-time queues is to be able to maintain a good quality-of-service even in the event of system noise.

In the conformance test, ID3D12CommandQueue::ExecuteCommandList will be deemed to be invoked after it returns – a QueryPerformanceCounter call will be made immediately after.

To measure when the GPU begin execution, a timestamp query will be placed at the very beginning of the command list. ID3D12CommandQueue::GetClockCalibration will be used to correlate the CPU QPC value and the GPU timestamp value.

Business Justification: The efficiency and latency improvements are required for Windows Holographic scenarios.

Device.Graphics.WDDM22.AdapterRender.D3D12Core.FastPSOComplies

Applies to

Windows 11 Client x64

Description:

D3D12 will add a flag to CreatePipelineState and shader creation methods, which indicates that the IHV driver should forego optimization during compilation of the relevant shaders, in order to achieve a quicker completion time.

Business Justification:

D3D9On12 and D3D11On12 are unable to invoke the driver shader compiler until a full set of state is known at draw-time. To continue making forward progress, these layers need a PSO as quickly as possible, and will replace it with an optimized compilation at a later time.

Device.Graphics.WDDM22.AdapterRender.D3D12Core.ExsistingSysMemHeaps

Applies to

Windows 11 Client x64

Description:

Drivers have already been required to interop with system memory allocated externally from the driver, using cross-adapter features. Drivers must now interop with all system memory in the same way, even when the driver allocates system memory itself. The runtime will use system memory features like write-watch without the driver's explicit awareness. The driver must also report a cap which indicates whether the GPU honors system memory page settings, like read-only.

Business Justification:

D3D12 tools must use write-watch memory to improve their performance.

Device.Graphics.WDDM22.AdapterRender.D3D12.WindowsHolographic

Device.Graphics.WDDM22.AdapterRender.D3D12.WindowsHolographic.CoreRequirements

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

Windows Holographic (WH) is an optional feature, however to support WH, graphics drivers must implement D3D12. In addition to D3D12 implementation, either the driver must support WDDM 2.2 or the driver must be on WDDM 2.0 with Graphics Preemption and Compute Preemption support:

- `DXGI_GRAPHICS_PREEMPTION_TRIANGLE_BOUNDARY`
- `DXGI_COMPUTE_PREEMPTION_THREAD_GROUP_BOUNDARY`

Relevant WDDM 2.2 features required for WH support include:

- `RealtimeCommandQueues`
- `PeriodicFrameNotification`
- `Late Latch`

Each feature is described separately in sections above.

Business justification: See business justification for each above feature in previous sections.

Device.Graphics.WDDM22.AdapterRender.D3D12.DepthBoundTest

Device.Graphics.WDDM22.AdapterRender.D3D12.DepthBoundTest.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

In implementing D3D12 in WDDM 2.2, drivers have the option to support Depth Bound Testing. This feature allows a programmer to discard pixels (or samples) based on a minimum and maximum depth value as compared against the currently stored depth value (not the one output by the PS under testing).

This API will be exposed as a PSO enable/disable toggle in order to support hardware that sets depth buffer read state in the PSO, and Command List method in order to support dynamically changing the Depth Bounds. It will allow

the setting of two floating point values: a minimum floating point depth value and a maximum floating point depth value.

Business Justification: This supports Xbox/Windows D3D parity.

Device.Graphics.WDDM22.AdapterRender.D3D12.ASTC

Device.Graphics.WDDM22.AdapterRender.D3D12.ASTC.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

In implementing D3D12 in WDDM 2.2, drivers have the option to support for ASTC texture formats.

Business Justification: This feature may be critical for platform competitiveness.

Device.Graphics.WDDM22.AdapterRender.D3D12.StereoPrimitives

Device.Graphics.WDDM22.AdapterRender.D3D12.StereoPrimitives.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

In implementing D3D12 in WDDM 2.2, drivers have the option to support stereo primitives. This feature provides ability for shader before rasterizer to output, in addition to usual xyzw position, an additional x value, allowing two projections translated on the x axis to be output from a single shader invocation.

Business Justification: Beneficial to VR scenarios, platform rendering efficiency.

Device.Graphics.WDDM22.AdapterRender.D3D12.MVPMRTArrayIndexRouting

Device.Graphics.WDDM22.AdapterRender.D3D12.MVPMRTArrayIndexRouting.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented**Description:**

In implementing D3D12 in WDDM 2.2, drivers have the option to support Multi-Viewport and Multi-RT-Array-Index Routing. This feature provides ability for shader before rasterizer to route output geometry to specific viewport(s) and RTArray index(es).

Business Justification:

Beneficial to VR scenarios, platform rendering efficiency.

Device.Graphics.WDDM22.AdapterRender.D3D12.ProgrammableMSAA

Device.Graphics.WDDM22.AdapterRender.D3D12.ProgrammableMSAA.CoreRequirements

Applies to	Windows 11 Client x64

Terms: If-Implemented**Description:**

In implementing D3D12 in WDDM 2.2, drivers have the option to support Programmable MSAA. This feature provides ability for application to select MSAA sample patterns.

Business Justification:

Beneficial to console parity, customer request, platform completeness.

Device.Graphics.WDDM22.AdapterRender.D3D12.9on12

Device.Graphics.WDDM22.AdapterRender.D3D12.9on12.FloatingPointBehavior

Applies to	Windows 11 Client x64

Terms: If-Implemented**Description:**

In implementing D3D12 in WDDM 2.2, drivers have the option to support 9on12 Floating Point Behavior. This feature is a method for asking drivers to tweak shader arithmetic behavior in ways to be compatible with legacy D3D9 applications.

Business Justification: Application compatibility.

Device.Graphics.WDDM22.AdapterRender.D3D11D3D12.SystemShaderCache

This feature is optional for D3D11 and D3D12, however, if a driver chooses to implement this feature for either D3D version the feature must also be implemented for the other D3D version.

Device.Graphics.WDDM22.AdapterRender.D3D11D3D12.SystemShaderCache.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

The D3D11 and D3D12 runtimes will provide access to a system shader cache infrastructure to the IHV driver to leverage. If a D3D11 driver opts to take advantage of this, they may leverage the provided callbacks with no additional requirements. If a D3D12 driver opts to take advantage of this, they must ensure that they only use the callbacks in response to appropriate DDI invocations.

For D3D12, the driver may only invoke the shader cache callbacks in response to the `pfnCreatePipelineState` DDI, or the `pfnCreateCommandSignature` DDI. The callbacks must be invoked on the same thread that called into the driver. Additionally, when called in response to `pfnCreatePipelineState`, the driver must provide the `HRTPIPELINESTATE` handle that was passed to the driver.

Business Justification: Having a shader cache improves application startup time.

Device.Graphics.WDDM22.AdapterRender.D3D12.DXILCore.ShaderModel60

Device.Graphics.WDDM22.AdapterRender.D3D12.DXILCore.ShaderModel60.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

Shader Model 6 is a new shader model that may be supported by drivers for D3D12. The major innovation here is the support of a new program representation, which also allows for the optional support for wave intrinsic functions and 64-bit int values. This feature is optional for WDDM 2.2 D3D12 drivers however if implementing, drivers must implement the following requirement.

To declare support for the feature, a driver must enumerate Shader Model 6 or higher as the highest supported shader model (`D3D12DDI_SHADER_MODEL_6_0_RELEASE_0011`). It should then accept shaders with major version 6, which will be encoded in DXIL bitcode rather than DXBC bytecode.

Business Justification:

Shader Model 6 modernizes the program representation and aligns it with LLVM, a popular open source toolchain that provides a wealth of supporting resources and which allows developers to familiarize themselves faster and reuse expertise from other compiler projects. This will lead to shorter cycles to deliver innovation as well as opening the platform to third-party tooling scenarios, feature and quality contributions from the development community, and partnering with academic research communities.

Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.WaveOps

Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.WaveOps.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

If a driver chooses to implement WaveOps it must implement ShaderModel6 outlined in the above section. In addition, to declare support for wave intrinsics, a driver should set the WaveOps member of the D3D12DDI_SHADER_CAPS_0012 structure to TRUE and initialize the WaveLaneCountMin and WaveLaneCountMax according to its capabilities, with a minimum value of at least 4 and a maximum value of at least its minimum. It should then process built-in functions related to wave operations as per their specification.

Business Justification:

The wave intrinsics requirement, if implemented, allows shader authors to exploit intra-wave communication, with guarantees around data visibility and synchronization that lead to better hardware utilization and improved performance. This leads to more competitive games and better user experiences.

Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.Int64

Device.Graphics.WDDM22.AdapterRender.D3D12.DXIL.Int64.CoreRequirement

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description:

If a driver chooses to implement Int64 it must implement ShaderModel6 outlined in the above section. In addition, to declare support for 64-bit int operations, a driver should set the Int64Ops member of the D3D12DDI_SHADER_CAPS_0015 structure to TRUE. It should then process the use of 64-bit signed and unsigned values in intrinsics and arithmetic opcodes as per their specification.

Business Justification:

The 64-bit int requirement, if implemented, allows shader authors to simplify code dealing with 64-bit int quantities natively, allowing improved hardware utilization and improved performance. Again, this can be used to provide better performance for games and some scientific applications.

Device.Graphics.WDDM23

WDDM 2.3 is an optional version level. If a driver implements WDDM 2.3, then it must implement all of the WDDM 2.3 Requirements.

Device.Graphics.WDDM23.AdapterRender.D3D12.DXILCore.ShaderModel61

Device.Graphics.WDDM23.AdapterRender.D3D12.DXILCore.ShaderModel61

Device.Graphics.WDDM23.AdapterRender.D3D12.DXILCore.ShaderModel61.CoreRequirementment

Shader Model 6.1 feature allowing user control on denorm arithmetic operations.

Applies to	Windows 11 Client x64

Description

Shader Model 6.1 is a new shader model that may be supported by drivers for D3D12. The major update in this version is supporting barycentrics interpolation and view instancing. This feature is optional for WDDM 2.3 D3D12 drivers however if implementing, drivers must implement the following requirement.

To declare support for the feature, a driver must enumerate Shader Model 6.1 or higher as the highest supported shader model. It should then accept shaders with major version 6, and minor version 1. Just like shader model 6.0, this shader model uses DXIL rather than DXBC for bytecode.

Device.Graphics.WDDM23.AdapterRender.D3D12.ViewInstancing

Device.Graphics.WDDM23.AdapterRender.D3D12.ViewInstancing

Device.Graphics.WDDM23.AdapterRender.D3D12.ViewInstancing.CoreRequirementment

Optional feature for Shader Model 6.1 to support view instancing

Applies to	Windows 11 Client x64

Description

If a driver chooses to implement view instancing it must implement ShaderModel6.1 outlined in the above section. In addition, to declare support for view instancing mode, a driver should set the ViewInstancingTier member of D3D12DDI_D3D12_OPTIONS_DATA_0033.

Device.Graphics.WDDM23.AdapterRender.D3D12.Video

WDDM 2.3 and later drivers that support video must implement these requirements both from D3D12.

Device.Graphics.WDDM23.AdapterRender.D3D12.Video.CoreRequirement

Required feature for WDDM2.3 D3D12.

Applies to	Windows 11 Client x64

Description

WDDM D3D12 drivers that support video are required to support the latest Video Feature Version and associated API and DDI for the WDDM version.

Device.Graphics.WDDM23.AdapterRender.D3D12.DepthBoundsTest

Device.Graphics.WDDM23.AdapterRender.D3D12.DepthBoundsTest.CoreRequirement

Applies to	Windows 11 Client x64

Description

Details of this requirement are in the “D3D12 – Depth Bounds Test – Unified Spec” doc.

Device.Graphics.WDDM23.AdapterRender.D3D12.ProgrammableMSAA

Device.Graphics.WDDM23.AdapterRender.D3D12.ProgrammableMSAA.CoreRequirement

Applies to	Windows 11 Client x64

Description

Details of this requirement are in the D3D12 programmable MSAA specification.

Device.Graphics.WDDM24

WDDM 2.4 is an optional version level. If a driver implements WDDM 2.4, then it must implement all of the D3D11 and D3D12 WDDM 2.4 Requirements.

Device.Graphics.WDDM24.CoreRequirements

WDDM 2.4 drivers must support these requirements and run the security testing successfully for the timeframe of the testing.

Required feature for WDDM2.4 D3D9-D3D12.

Applies to	Windows 11 Client x64

Description

Graphics drivers that implement WDDM 2.4 must be secure against agent-in-the-middle fuzzing of their private data structures in UMD to KMD communications through the DDI to ensure a secure paravirtualization interface.

Device.Graphics.WDDM24.Render

Device.Graphics.WDDM24.Render.D3D11VideoDecoding

Applies to	Windows 11 Client x64

Description

Details of this feature are described in the D3D11_AsyncNegotiateCryptoSessionKeyExchange and D3D11 & D3D12 Video Decode Histogram specifications.

Device.Graphics.WDDM24.Render

WDDM 2.4 drivers must support these requirements from D3D9 to D3D12.

Device.Graphics.WDDM24.Render.CrossAdapterResource

Required feature for WDDM2.4 D3D9-D3D12.

Applies to	Windows 11 Client x64

Description

Graphics drivers for all UMDs (D3D9-12) must meet all Cross Adapter Resource support requirements defined in the Windows Hybrid Graphics DDI Specifications, and set the KMD DXGK_VIDMMCAPS::CrossAdapterResource cap to true.

Device.Graphics.WDDM24.Render.D3D11.VideoDecoding

Required feature for WDDM2.4 D3D12.

Applies to	Windows 11 Client x64

Description

WDDM 2.4 D3D11 drivers that support video are required to support the latest WDDM 2.4 video DDI.

Cross-API sharing of protected resources with D3D12.

Device.Graphics.WDDM24.Render.D3D12ResourceSerialization

Required feature for WDDM2.4 D3D12, even if hardware does not support heap serialization.

Applies to	Windows 11 Client x64

Description

The existing resource state barrier spec is changed to say, "Implicitly promoted resource barrier states on non-simultaneous-access textures also decay, as long as the implicitly promoted state hasn't accumulated a GPU write state bit."

Device.Graphics.WDDM24.Render.D3D12VideoDecoding

Required feature for WDDM2.4 D3D12.

Applies to	Windows 11 Client x64

Description

WDDM 2.4 D3D12 drivers that support video are required to support the latest WDDM 2.4 Video Feature Version and associated API and DDI.

Device.Graphics.AdapterRender.D3D11WDDM24

WDDM 2.4 drivers must support eFSE

Device.Graphics.AdapterRender.D3D11WDDM24.eFSE

WDDM 2.4 drivers must implement eFSE DDIs

Applies to	Windows 11 Client x64

Description

The driver must extend existing support of the pfnRotateResourceIdentities DDI, as defined in the DXGI specification, with sufficient capability to rotate resources with heterogeneous properties as outlined in the WDDM 2.4 eFSE specification.

Device.Graphics.WDDM24.AdapterRender.D3D11.SharedTexture

WDDM 2.4 drivers must support these requirements both from D3D11 and D3D12.

Device.Graphics.WDDM24.AdapterRender.D3D11.SharedTexture.CoreRequirement

Required feature for WDDM2.4 D3D11.

Applies to	Windows 11 Client x64

Description

Hardware partners must start understanding better how the D3D12 resource binding design, D3D12 memory model, and OS security designs remove the flexibility available to D3D11 drivers. This significantly impacts SoC designs where texture memory layouts aren't understood orthogonally from all system processors.

Device.Graphics.WDDM24.AdapterRender.D3D12.ContentProtection

WDDM 2.4 drivers that support HWDRM must implement protected resources with D3D12.

Device.Graphics.WDDM24.AdapterRender.D3D12.ContentProtection.CoreRequirement

Required feature for WDDM2.4 D3D12.

Applies to	Windows 11 Client x64

Description

WDDM 2.4 D3D11 drivers that support HWDRM are required to support the latest WDDM 2.4 Protected Resource Feature Version and associated API and DDI.

Device.Graphics.WDDM24.AdapterRender.D3D12.DXILCore.ShaderModel62

Device.Graphics.WDDM24.AdapterRender.D3D12.DXILCore.ShaderModel62

Device.Graphics.WDDM24.AdapterRender.D3D12.DXILCore.ShaderModel62.CoreRequirement

Shader Model 6.2 feature allowing user control on denorm arithmetic operations.

Applies to	Windows 11 Client x64

Description

Shader Model 6.2 is a new shader model that may be supported by drivers for D3D12. The major update in this version is to provide user option for denormal number operations, and other non-required feature such as float16/int16. This feature is optional for WDDM 2.4 D3D12 drivers. However if implementing, drivers must implement the following requirement.

To declare support for the feature, a driver must enumerate Shader Model 6.2 or higher as the highest supported shader model. It should then accept shaders with major version 6, and minor version 2. Just like previous shader model 6 version, this shader model uses DXIL rather than DXBC for bytecode.

Device.Graphics.WDDM24.AdapterRender.D3D12.Native16Bits

Device.Graphics.WDDM24.AdapterRender.D3D12.FP16

Device.Graphics.WDDM24.AdapterRender.D3D12.Native16Bits.CoreRequirement

Optional feature for Shader Model 6.2 to introduce fp16 type

Applies to	Windows 11 Client x64

Description

If a driver chooses to implement FP16 it must implement ShaderModel6.2 outlined in the above section. In addition, to declare support for FP16 mode, a driver should set the NativeHalf member of the structure to TRUE. It should then process the use of fp16/int16 mode operations as per their specification.

Basic arithmetic operations on FP16 is supported, but not atomic operations.

Device.Graphics.WDDM24.AdapterRender.D3D12.PeerToPeerLDAAtomics

WDDM 2.4 drivers must divulge whether the hardware supports this optional capability.

Device.Graphics.WDDM24.AdapterRender.D3D12.PeerToPeerLDAAtomics.CoreRequirement

Optional feature for WDDM2.4 D3D12.

Applies to	Windows 11 Client x64

Description

Discrete graphics hardware must support the following atomic memory transactions over a bus that connects the two linked adapters:

- atomic_and
- atomic_cmp_store

- atomic_iadd
- atomic_imax
- atomic_imin
- atomic_or
- atomic_umax
- atomic_umin
- atomic_xor

Device.Graphics.WDDM24.AdapterRender.D3D12.SharedTexture

WDDM 2.4 drivers must support these requirements both from D3D11 and D3D12.

Device.Graphics.WDDM24.AdapterRender.D3D12.SharedTexture.CoreRequirement

Required feature for WDDM2.4 D3D12.

Applies to	Windows 11 Client x64

Description

Hardware partners must start understanding better how the D3D12 resource binding design, D3D12 memory model, and OS security designs remove the flexibility available to D3D11 drivers. This significantly impacts SoC designs where texture memory layouts aren't understood orthogonally from all system processors.

Device.Graphics.WDDM24.Paravirtualization

WDDM 2.4 drivers must support GPU paravirtualization for DirectX APIs.

Device.Graphics.WDDM24.Paravirtualization.CoreRequirement

WDDM 2.4 drivers for DirectX APIs must implement paravirtualization DDIs and be capable of properly deploying and functioning in a virtualized environment.

Applies to	Windows 11 Client x64

Description

WDDM 2.4 drivers for DirectX must meet all virtualization requirements defined in the WDDM 2.4 specifications. The virtualization requirements are outlined below:

- The driver must set the DXGK_VIDMMCAPS::ParavirtualizationSupported capability.
- The driver must conform to the DDI changes and driver installation constraints described in the WDDM 2.4 specifications.
- When running in a virtualized guest, the UMD must not use any process context related data in the private driver data.

- When running in a virtualized guest, the UMD cannot share memory with the KMD in the host.
- When running in a virtualized guest, the UMD must only access the registry with the D3DDDI_ADAPTERCALLBACKS::pfnQueryAdapterInfoCb2 callback or the D3DKMTQueryAdapterInfo function.
- The KMD must support the new IOMMU-based GPU isolation DDI and operate properly within such an environment.

These requirements apply to DirectX APIs: DX9, DX10, DX11, DX12.

Further, DX10, DX11, and DX12 must support a high-security mode where the driver can operate without escapes, per the WDDM 2.4 specifications.

Device.Graphics.WDDM25

WDDM 2.5 is an optional version level. If a driver implements WDDM 2.5, then it must implement all of the WDDM 2.5 Requirements except those explicitly marked as optional.

Device.Graphics.WDDM25.AdapterRender.D3D12.Raytracing

Device.Graphics.WDDM25.AdapterRender.D3D12.Raytracing

Device.Graphics.WDDM25.AdapterRender.D3D12.DXILCore.ShaderModel63

Optional feature. Enables DirectX12 RayTracing.

Applies to	Windows 11 Client x64

Description

Shader model 6.3 adds support for DirectX12 Raytracing (DXR). This includes new intrinsics for casting rays into acceleration data structures and processing the results. It also includes official HLSL compiled DXIL libraries and linking of shaders from these libraries, which has applications beyond raytracing. See the DX12 Raytracing documentation (Direct3D 12 Raytracing - Win32 apps | Microsoft Docs)

To declare support for these features, a driver must enumerate Shader Model 6.3 or higher as the highest supported shader model. It should then accept shaders with major version 6, and minor version 3. The shader model 6.3 HLK tests must also pass.

Device.Graphics.WDDM25.AdapterRender.D3D12.Raytracing.CoreRequirement

D3D12 devices can optionally expose raytracing support.

Applies to	Windows 11 Client x64

Description

Details of this optional feature are in: D3D12 Raytracing Functional Spec.

Device.Graphics.WDDM25.Display.Brightness3

WDDM 2.5 drivers must report whether the hardware supports this optional capability.

Device.Graphics.WDDM25.Display.Brightness3.InstantaneousChange

Optional feature for WDDM 2.5

Applies to	Windows 11 Client x64

Description

Instantaneous brightness changes must be supported. A SetBrightness call with TransitionTimeMs field set to 0 must succeed and immediately set the brightness to the target BrightnessMillinits level. The next GetBrightness call must return the BrightnessMillinits level that was just set.

More details of this optional feature are in the Brightness3 DDI Spec.

Device.Graphics.WDDM25.Display.Brightness3.LinearRamping

Optional feature for WDDM 2.5

Applies to	Windows 11 Client x64

Description

Ramped brightness changes must be linear over time. A SetBrightness call with TransitionTimeMs set to a non-zero value must succeed and perform a linear transition from the current brightness level to the target BrightnessMillinits level over the length of TransitionTimeMs.

More details of this optional feature are in the Brightness3 DDI Spec.

Device.Graphics.WDDM25.Display.Brightness3.PreferredMaximumBrightness

Optional feature for WDDM 2.5

Applies to	Windows 11 Client x64

Description

PreferredMaximumBrightness in DXGK_BRIGHNTESS_GET_NIT_RANGES_OUT must be set to a value in millinits that is less than or equal to the highest brightness level in the normal ranges.

More details of this optional feature are in the Brightness3 DDI Spec.

Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayDithering

WDDM 2.5 drivers must report whether the hardware supports this optional capability.

Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayDithering.CoreRequirements

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

WDDM 2.5 graphics drivers must report whether display dithering is supported on a per-monitor link basis. Dithering will be applied if supported by the driver whenever the surface format pixel depth is greater than the wire format pixel depth. In summary, drivers that wish to support HDR over an 8-bit wire format **MUST** support DisplayDithering; in the absence of dithering, the OS will only support HDR over a 10-bit wire format.

Additional details:

- Drivers must determine when dithering should be applied based on whether it will provide a visual quality benefit to the active configuration.
- Drivers must not report dithering capabilities if they intend to not apply dithering.

Device.Graphics.WDDM25.Display.HighDynamicRange.SDRWhiteLevel

WDDM 2.5 drivers must report whether the hardware supports this optional capability when on hardware shipped in 2019 or later.

Device.Graphics.WDDM25.Display.HighDynamicRange.SDRWhiteLevel.CoreRequirement

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

WDDM2.5 drivers that implement HDR **MUST** support the new SDR White Level value within DXGK_COLORIMETRY2 as part of DxgkDdiSetTargetAdjustedColorimetry2. Drivers should cache the SDR While Level value as with similar parameters of DXGK_COLORIMETRY2 and must apply the updated value within a reasonable timeframe, e.g. ~2 frames. Drivers must also apply this SDR White Value to the hardware cursor whenever possible. In the case that a driver is unable to support an SDR White Level value for hardware cursor, the driver must fail the DxgkDdiSetPointerShape() call, in which case the OS will fall back to software cursor.

Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayColorspaceTransform

WDDM 2.5 drivers must report whether the hardware supports this optional capability when on hardware shipped in 2019 or later.

Device.Graphics.WDDM25.Display.HighDynamicRange.DisplayColorspaceTransform.CoreRequirement

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

Drivers that support the DisplayColorSpaceTransform DDI must conform to the DisplayColorSpaceTransform DDI specifications.

Drivers that support this DDI must compose the following four logical colorspace conversation matrices, in this order

1. Source RGB to XYZ
2. XYZ to XYZ adjustment
3. Scaling
4. XYZ to Target RGB

If a driver reports that WideColor colorspace transforms are supported, this stage must also support the following requirements

- Input color values within the SDR BT.2020 colorspace, as encoded in scRGB, must be preserved without clamping. Ideally, input color values within the range of scRGB [-2.0, 2.0] are preserved without clamping. Clamping to [0, 1] only occurs after the matrix has been applied.

If the driver reports that HighColor colorspace transforms are supported, this stage must support the following requirements:

- Input color values within the HDR10 colorspace (BT.2020, ST.2084 10k nits max), as encoded in scRGB, must be preserved without clamping. Ideally, input color values within the range of scRGB [-128, 256] are preserved without clamping. Clamping to [0, 1] only occurs after the matrix has been applied.

Drivers must support scale/scale-by-multiplier - though this may be fully implemented within the XYZ to XYZ adjustment above.

Drivers shall not perform an explicit color management to the physical primaries or physical gamut of the display. The OS controls this explicitly via the DDI.

Drivers shall not persist any internal colorspace transform state that overlaps with state defined by these DDIs.

Drivers must also conform to the following requirements:

- It must be supported regardless of pixel clock, timings and wire-format combinations
- Changes to the transforms must be performed by the hardware or driver during the vertical blanking period
- Changing the transforms must not cause any visual artifacts, other than the programmed transform
- If an identity mapping is used for the colorspace transforms, there must be no visual artifacts when enabling or disabling the use of the transform hardware

Device.Graphics.WDDM25.Display.HighDynamicRange.HDRPixelFormatSupport

WDDM 2.5 drivers must report whether the hardware supports this optional capability.

Device.Graphics.WDDM25.Display.HighDynamicRange.HDRPixelFormatSupport.CoreRequirement

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

WDDM 2.5 drivers that support HDR must include support for the pixel formats described in the HDR Pixel Format Support specification. Drivers must not report support for ARGB10HDR pixel formats unless the IHV has worked directly with Microsoft to ensure a viable solution for the applicable device.

Drivers that support HDR must report support for either HDRFP16ScanoutSupport OR HDRARGB10ScanoutSupport, but not both.

Device.Graphics.WDDM25.Display.HighDynamicRange.Brightness3

WDDM 2.5 drivers must report whether the hardware supports this optional capability.

Device.Graphics.WDDM25.Display.HighDynamicRange.Brightness3.CoreRequirement

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

Drivers that support Brightness3 must implement the support for the following methods:

- SetBrightness
- GetBrightness
- GetBrightnessCaps
- GetNitRanges
- SetBacklightOptimization

Any HDR display that offers support for brightness control (e.g., integrated panels) MUST implement Brightness3 with calibrated nits. This nits-based brightness is required for HDR content to map to the display with the best quality. The specific requirements of Brightness3 are defined in the “HDR Brightness Compensation” DDI specification.

Device.Graphics.WDDM25.Display.HighDynamicRange.DirectFlipSupport

WDDM 2.5 drivers must report whether the hardware supports this optional capability when on hardware shipped in 2019 or later.

Device.Graphics.WDDM25.Display.HighDynamicRange.DirectFlipSupport.CoreRequirement

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

Drivers that support HDR must include DirectFlip support for the following formats:

- RGBA8888 (SDR)
- FP16 (HDR)
- RGBA1010102 (HDR)
- RGBA1010102 (SDR)

Device.Graphics.WDDM25.Display.HighDynamicRange.NeutralColorProcessingControl

WDDM 2.5 drivers must report whether the hardware supports this optional capability.

Device.Graphics.WDDM25.Display.HighDynamicRange.NeutralColorProcessingControl.CoreRequirement

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

Note that support for this feature only applies to complete WDDM drivers, e.g., drivers which consist of both rendering and display functions. This DDI enables control for all visual post processing performed within a driver that the OS has no knowledge of (e.g., color transforms using proprietary IHV APIs)

Drivers that support this feature must set the appropriate DXGK_MODE_BEHAVIOR_FLAGS values corresponding with the desired behavior. Drivers are not required to virtualize calls to this DDI and should apply requested changes immediately.

Device.Graphics.WDDM25.SRVOnlyTiledResourceTier3

WDDM 2.5 drivers must report whether the hardware supports this optional capability.

Device.Graphics.WDDM25.SRVOnlyTiledResourceTier3.CoreRequirement

Optional feature for WDDM2.5

Applies to	Windows 11 Client x64

Description

WDDM 2.5 drivers that support SRV-Only Tiled Resource Tier 3 must conform to the specification and pass the corresponding HLK tests. The specification and tests mimic Tiled Resource Tier 3 support; but are more constrained. Only shader-resource-view (SRV) access and data copying operations must conform to the original Tiled Resource Tier 3 design.

Device.Graphics.WDDM25.OneCoreUAP

WDDM 2.5 drivers must support OneCoreUAP

Device.Graphics.WDDM25.OneCoreUAP.CoreRequirement

WDDM 2.5 drivers must be able to load and run DirectX workloads on OneCoreUAP-based devices, both natively and under paravirtualization.

Applies to	Windows 11 Client x64

Description

Details of this requirement will be available in a WDDM 2.5 specification. The requirement is outlined below:

- DX11 and DX12 UMDs must be compiled against the OneCoreUAP libs and function correctly on OneCoreUAP-based OSes.
- The KMD must be compiled against the OneCoreUAP libs and function correctly on OneCoreUAP-based OSes.
- The driver must support paravirtualization on OneCoreUAP-based OSes, as the host and as the guest, in compatibility mode.

Device.Graphics.WDDM26

WDDM 2.6 is an optional version level. If a driver implements WDDM 2.6, then it must implement all of the WDDM 2.6 Requirements except those explicitly marked as optional.

Device.Graphics.WDDM26.OneCore

WDDM 2.6 drivers must support OneCore.

Device.Graphics.WDDM26.OneCore.CoreRequirement

WDDM 2.6 drivers must be able to load and run DirectX workloads on OneCore-based devices.

Applies to	Windows 11 Client x64

Description

Details of this requirement are available in a WDDM 2.6 specification. The requirement is outlined below:

- DX12 UMDs must be compiled against the OneCore libs and function correctly on OneCore-based OSes.
- The KMD must be compiled against the OneCore libs and function correctly on OneCore-based OSes.

Device.Graphics.WDDM26.Containers

WDDM 2.6 drivers must support running inside of Windows Server Containers as well as Hyper-V Containers.

Device.Graphics.WDDM26.Containers.CoreRequirement

WDDM 2.6 drivers must be able to load and run DirectX 11 and DirectX 12 workloads within Windows containers.

Applies to	Windows 11 Client x64

Description

Details of this requirement are available in a WDDM 2.6 specification.

Drivers must function correctly while running inside of a container environment where they are isolated from the host file system and registry, and must ensure that they use the appropriate kernel services to access any necessary data.

Device.Graphics.WDDM26.D3D12.Metacommands

This feature enables D3D applications to opt into using a better-accelerated version of known algorithms on the target IHV hardware.

Device.Graphics.WDDM26.D3D12.Metacommands.GpuDecompression

Leverage the GPU to perform decompression, freeing up the CPU for other things. DirectStorage leverages this metacommand to improve load times in games.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Compressed data is passed to the metacommand which gets decompressed by the driver using the GPU. This gives each IHV a chance to craft optimal/high performance implementations of GPU based decompression tailored to their hardware design.

Device.Graphics.WDDM26.D3D12.Metacommands.MachineLearning

IHV's can optionally expose hardware accelerated versions of specific known algorithms to applications utilizing Windows ML and DirectML.

Applies to	Windows 11 Client x64

Description

IHV's can expose hardware accelerated versions of specific known algorithms to applications utilizing Windows ML and DirectML for machine learning scenarios by exposing DirectML meta commands. The implementable set of DirectML meta commands are detailed in the [document](#). This document is available to hardware vendors by request to askwindowxml@microsoft.com.

Device.Graphics.WDDM26.D3D12.VariableRateShading

This feature enables D3D applications to opt into rendering performance benefits available on the target IHV hardware.

Device.Graphics.WDDM26.D3D12.VariableRateShading.CoreRequirement

D3D12 devices can optionally expose variable rate shading support.

Applies to	Windows 11 Client x64

Description

Details of this optional feature are in the D3D12 Variable Rate Shading spec.

VRS spec <https://github.com/microsoft/DirectX-Specs/blob/master/d3d/VariableRateShading.md>

Device.Graphics.WDDM26.HardwareMotionVectorExtraction

WDDM 2.6 drivers must report whether the hardware supports this optional capability.

Device.Graphics.WDDM26.HardwareMotionVectorExtraction.CoreRequirement

Optional feature for WDDM2.6

Applies to	Windows 11 Client x64

Description

This requirement will become effective from the Windows release after February, 2019.

WDDM 2.6 drivers that support Hardware Motion Vector Extraction must conform to the HardwareMotionVectorExtraction specification and pass the corresponding HLK tests.

Device.Graphics.WDDM27

WDDM 2.7 is an optional version level. If a driver implements WDDM 2.7, then it must implement all of the WDDM 2.7 Requirements except those explicitly marked as optional.

Device.Graphics.WDDM27.D3D12.SamplerFeedback

This feature enables D3D applications to opt into a graphics feature available on the target IHV hardware.

Device.Graphics.WDDM27.D3D12.SamplerFeedback.CoreRequirement

D3D12 devices can optionally expose sampler feedback support.

Applies to	Windows 11 Client x64

Description

Details of this optional feature are in the D3D12 Sampler Feedback spec.

Sampler feedback https://dev.azure.com/cga-exchange/docs/_git/docs?path=%2Fd3d%2FSamplerFeedback.md&version=GBmaster&fullScreen=true

Device.Graphics.WDDM27.D3D12.MeshShaders

This feature enables D3D applications to opt into a graphics feature on the target IHV hardware.

Device.Graphics.WDDM27.D3D12.MeshShaders.CoreRequirement

D3D12 devices can optionally expose mesh shader support.

Applies to	Windows 11 Client x64

Description

This feature adds new stages to the Direct3D graphics pipeline. Details of this optional feature are in the D3D12 Mesh Shaders spec.

Mesh shaders https://dev.azure.com/cga-exchange/_git/docs? a=preview&path=%2Fd3d%2FMeshShader.md&version=GBmaster&fullScreen=true

Device.Graphics.WDDM27.D3D12.TrackedWorkloads

This feature enables D3D applications to opt into power-saving features on the target IHV hardware.

Device.Graphics.WDDM27.D3D12.TrackedWorkloads.CoreRequirement

D3D12 devices can optionally expose tracked workloads support.

Applies to	Windows 11 Client x64

Description

Details of this optional feature are in the D3D12 Tracked Workloads spec.

Tracked workloads https://dev.azure.com/cga-exchange/docs/_git/docs?path=%2Fd3d%2FDynamicGoalBasedGPUPowerManagement.md&version=GBmaster&fullScreen=true

Device.Graphics.WDDM27.AdapterRender.D3D12.ContentProtection

This D3D12 feature enables optional protected resource support for video operations.

Device.Graphics.WDDM27.AdapterRender.D3D12.ContentProtection.CoreRequirement

D3D12 devices can optionally expose protected resource support.

Applies to	Windows 11 Client x64

Description

Details of this optional feature are in the D3D12 Protected Resources spec.

Content protection https://dev.azure.com/cga-exchange/docs/_git/docs?path=%2Fd3d%2FD3D12_Video_ProtectedResourceSupport.md&version=GBmaster&fullScreen=true

Device.Graphics.WDDM27.ParavirtualizationNonDirectX

WDDM 2.7 drivers must support GPU paravirtualization for non-DirectX APIs.

Device.Graphics.WDDM27.ParavirtualizationNonDirectX.CoreRequirement

WDDM 2.7 drivers for non-DirectX APIs must implement paravirtualization DDIs and be capable of properly deploying and functioning in virtualized and containerized environments.

Applies to	Windows 11 Client x64

Description

WDDM 2.7 drivers for OpenGL, OpenCL, Vulkan, and any IHV-specific graphics API must meet all virtualization and containers requirements previously defined for DirectX APIs in Device.Graphics.WDDM24.Paravirtualization and Device.Graphics.WDDM26.Containers. Details of these requirements are available in the corresponding WDDM specs. See also the following documentation:

- <https://docs.microsoft.com/en-us/windows-hardware/drivers/display/container-non-dx>

Additionally, user mode drivers and kernel mode drivers for all graphics APIs must not rely on matched OS versions between the host and the guest. WDDM 2.7 and newer user mode drivers must be able to run in new guest OS versions on Windows 10, version 1809 and Windows Server, version 1809 hosts or later.	

Device.Graphics.WDDM27.Display.Diagnostics

WDDM 2.7 drivers must support the blackscreen diagnostics DDI

Device.Graphics.WDDM27.Display.Diagnostics.CoreRequirement

WDDM 2.7 drivers must implement the blackscreen diagnostics DDI and correctly demonstrate the detection of blackscreens in expected scenarios.

Applies to	Windows 11 Client x64

Description

Drivers implementing the whitebox and blackbox data collection DDIs enable the collection of useful telemetry in scenarios where users may experience a blackscreen. This DDI is required, but drivers are allowed to not report individual data points where such collection is not feasible.

Device.Graphics.WDDM27.Display.MipiDsi

WDDM 2.7 drivers must support the blackscreen diagnostics DDI

Device.Graphics.WDDM27.Display.MipiDsi.CoreRequirement

WDDM 2.7 drivers that choose to implement the optional MIPI/DSI DDIs should provide consistent support for messages defined by the DDI.

Applies to	Windows 11 Client x64

Description

Drivers may optionally choose to implement the MIPI/DSI DDIs which allow OEMs to send arbitrary MIPI/DSI commands to a MIPI-connected panel. If this DDI is implemented by a driver, it should support the full range of DSI messages defined in the DDI, including basic panel data collection.

Device.Graphics.WDDM28

WDDM 2.8 is an optional version level. If a driver implements WDDM 2.8, then it must implement all of the WDDM 2.8 Requirements except those explicitly marked as optional. A WDDM 2.8 driver must meet all requirements defined in the WDDM 2.8 Specification.

Device.Graphics.WDDM29

WDDM 2.9 is an optional version level. If a driver implements WDDM 2.9, then it must implement all of the WDDM 2.9 Requirements except those explicitly marked as optional. A WDDM 2.9 driver must meet all requirements defined in the WDDM 2.9 Specification.

Device.Graphics.WDDM29.Render.CoreRequirement

A WDDM 2.9 render driver must meet all requirements defined in the WDDM 2.9 Specification.

Applies to	Windows 11 Client x64

Description

In particular, drivers should implement the following (depending on hardware capabilities):

- If WDDM 2.9 driver supports DList, it must export the QueryDListForApplication2 function via their DList driver dll.

Device.Graphics.WDDM30

WDDM 3.0 is an optional version level. If a driver implements WDDM 3.0, then it must implement all of the WDDM 3.0 Requirements except those explicitly marked as optional. A WDDM 3.0 driver must meet all requirements defined in the WDDM 3.0 Specification.

Device.Graphics.WDDM30.AdapterRender.D3D12.DXILCore.ShaderModel66

Optional feature.

Applies to	Windows 11 Client x64

Description

Shader Model 6.6 adds floating point and 64-bit atomic operations, derivatives in compute, mesh and amplification shaders, dynamic resources, new pack and unpack datatypes and intrinsics, and a new WaveSize option.

- 64-bit integer and limited floating-point atomic operations were added by overloading the Interlocked* functions and methods used on group shared memory, raw buffer, and typed (RWBuffer/RWTexture) resources.

- Support for quad-based derivative operations to compute, mesh and amplification shaders were added. Previously, these operations were unavailable in these shader stages because they require computations on 2x2 quads of adjacent values. Shaders without this explicit structure can now use these operations.
- The new Dynamic resource feature introduces the ability to create resources from descriptors by directly indexing into the CBV_SRV_UAV heap or the Sampler heap. No root signature descriptor table mapping is required for this resource creation method, but new global root signature flags are used to indicate the use of each heap from the shader.
- New packed datatypes and intrinsics for processing of packed 8bit data such as colors (int8_t4_packed, uint8_t4_packed)
- Finally, Shader Model 6.6 introduces a new option that allows the shader author to specify a wave size that the shader is compatible with.

To declare support for these features, a driver must enumerate Shader Model 6.6 or higher as the highest supported shader model. It should then accept shaders with major version 6, and minor version 6. The shader model 6.6 HLK tests must also pass.

Device.Graphics.WDDM30.Render.CoreRequirement

Applies to	Windows 11 Client x64

In particular, drivers should implement the following (depending on hardware capabilities):

- If WDDM 3.0 driver is HybridIntegrated, it must support CrossAdapterResourceScanout driver capabilities.

Device.Graphics.WDDM30.DisplayRender.CrossAdapterScanOut

Details of this optional feature are in the Cross Adapter Resource Scan Out spec.

Applies to	Windows 11 Client x64

Device.Graphics.WDDM30.Render.SelfRefreshMemorySupport

WDDM 3.0 drivers must support VRAM Self Refresh Memory on Modern Standby systems.

Device.Graphics.WDDM30.Render.SelfRefreshMemorySupport.CoreRequirement

WDDM 3.0 drivers for GPUs with VRAM must support Self Refresh Memory on Modern Standby systems.

Applies to	Windows 11 Client x64

Description

Modern Standby is an important power saving capability for Windows 10. WDDM 3.0 drivers must support VRAM Self Refresh Memory for Modern Standby systems. Without Self Refresh and careful design choices, it's possible to lose

the instant-on and low power benefits that Modern Standby provides. See the following for more information on supporting GPUs with VRAM on Modern Standby systems:

- <https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/modern-standby-graphics>

Device.Graphics.WDDM30.Paravirtualization.BackwardsCompatvGpuSupport

WDDM 3.0 drivers must support backwards compatibility.

Device.Graphics.WDDM30.Paravirtualization.BackwardsCompatvGpuSupport.CoreRequirement

WDDM 3.0 for GPUs that support para-virtualization must also support backwards compatibility support for older VMs running on newer hosts.

Applies to	Windows 11 Client x64

Description

Backwards compatibility for vGPU is a key paravirtualized scenario to enable some emulator products on Windows 10. Starting in WDDM 3.0 a new test will be added to the HLK enforcing backwards compatibility for paravirtualized adapters, meaning vGPU must work for older VM/Container images running on a newer host OS.

Device.Graphics.WDDM30.WSL

WDDM 3.0 drivers must support running in a Linux guest environment through the Linux for Windows Subsystem.

Device.Graphics.WDDM30.WSL.CoreRequirement

WDDM 3.0 for GPUs that support para-virtualization must also support running in a Linux guest..

Applies to	Windows 11 Client x64

Description

WDDM 3.0 drivers will be required to run compute-based workloads in a Linux virtual machine running on a Windows host. This interface will be provided through our existing paravirtualization infrastructure and our Linux mode support, new tests will be introduced to the HLK to test this support through the Linux subsystem for Windows.

Device.Graphics.WDDM30.DisplayRender.FlexibleDisplayableSurfacePresentation

This is an optional feature.

WDDM 3.0 drivers may choose to support texture 2D creation via the D3D11 CreateResource DDI with flag D3DWDDM2_ODDI_RESOURCE_MISC_DISPLAYABLE_SURFACE, and if so, must support flexible presentation of such textures.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The ID3D11Device::CreateTexture2D API is updated to support a new D3D11_TEXTURE2D_DESC MiscFlag, D3D11_RESOURCE_MISC_SHARED_DISPLAYABLE, which when used, must be used in combination with both D3D11_RESOURCE_MISC_SHARED and D3D11_RESOURCE_MISC_SHARED_NTHANDLE flags. In such situation, the driver's D3D11 CreateResource DDI will receive MiscFlag values including D3DWDDM2_0DDI_RESOURCE_MISC_DISPLAYABLE_SURFACE and D3D10_DDI_RESOURCE_MISC_SHARED. Note that these DDI flag values are pre-existing, but this usage of them for this scenarios is new.

Such textures must be displayable for flexible presentation use. These textures are not required to have the same properties - for example, formats and sizes can differ, and these textures must be able to be displayed in arbitrary order ("out of order presentation"). Presentation will occur using the existing Present1 DDI, with its existing calling patterns.

Device.Graphics.WDDM30.Display.DisplayPortAux

WDDM 3.0 drivers must support the DisplayPort AUX DDI

Device.Graphics.WDDM30.Display.DisplayPortAux.CoreRequirement

WDDM 3.0 drivers must implement the DisplayPort AUX DDI for devices which expose an external DisplayPort target.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Implementing the new DisplayPort AUX channel DDI is essential for enabling telemetry and diagnostics around docking scenarios. The DisplayPort AUX channel messages should be handled without interference from other DDIs. Drivers should ensure that the DisplayPort bus is in the correct state for DPCD or I2C reads. A new HLK test will be added to ensure drivers can read DPCD registers over AUX and EDID using I2C over AUX.

Device.Graphics.WDDM30.Display.DisplayPortSbm

WDDM 3.0 drivers must support the DisplayPort SBM DDI

Device.Graphics.WDDM30.Display.DisplayPortSbm.CoreRequirement

WDDM 3.0 drivers must implement the DisplayPort SBM DDI for devices which expose an external DisplayPort target.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Implementing the new DisplayPort SBM DDI is essential for enabling telemetry and diagnostics around docking scenarios. The DisplayPort AUX channel messages should be handled without interference from other DDIs. Drivers should ensure that the DisplayPort bus is in the correct state for a message transaction to be sent (LINK_ADDRESS, ENUM_PATH_RESOURCES, QUERY_PAYLOAD, GET_MESSAGE_TRANSACTION_VERSION, REMOTE_DPCD_READ and REMOTE_I2C_READ). A new HLK test will be added to ensure drivers can read DPCD registers of every device along the path in topology of either a MST display or a SST display connected through a DP branch device/dock and EDID of the target display using message transactions.

Device.Graphics.WDDM32.Display.DisplayMux

Systems with multiple GPUs and an integrated display may choose to include a display mux. Drivers on these systems must support this functionality.

Device.Graphics.WDDM32.Display.DisplayMux.CoreRequirement

Requirement for drivers supporting systems with a display mux.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
System must pass all the device DisplayMux HLK tests, please see the "DisplayMux - Test and getting To Ship Quality" document for more details about the testing requirements along with the details of the light sensor device that is required to run the HLK tests.	

Device.Graphics.WDDM32.Display.DisplayMux.DynamicSwitching

Requirement for drivers supporting systems with a display mux.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
Systems that support a Display Mux device may also support OS-managed dynamic switching of which GPU is currently driving the display. This dynamic switching considers several factors to determine whether an application should trigger a change in which GPU is driving the display. Systems that support this functionality must meet the following requirements:	
The dedicated GPU driver can optionally support reporting a per-app dynamic switching preference to be considered in the dynamic switch decision. If the driver reports support for this DDI, the OS must be able to successfully query a preference from the associated dList DLL.	

Device.Imaging

Device.Imaging.Printer.Base

Device.Imaging.Printer.Base.applicationVerifier

Printer driver components must comply with Application Verifier test criteria.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

This requirement clarifies the use of existing WLK 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](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](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 11 Client 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

INF files must have correct syntax when PWG Raster rendering filter is included

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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 11 Client x64

Description

This requirement clarifies the use of existing WLK tests. The **optional** 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](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](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 11 Client x64

Description

This requirement clarifies the use of existing WLK 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.ColInstallers, 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](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](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 11 Client x64 Windows 11 Client ARM64

Description

This requirement clarifies the use of existing WLK 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](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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64

Windows 11 Client ARM64

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 11 Client x64

Windows 11 Client ARM64

Description

This requirement clarifies the **optional** use of the following existing WLK 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 **optional** 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](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](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](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](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 11 Client x64 Windows 11 Client ARM64

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.

Device.Imaging.Printer.Mobile

Device.Imaging.Printer.Mobile.PDL

Printers with **Mopria** or mobile print capability that support one of the following PDLs must conform to the relevant requirements for it.

Applies to	Windows 11 Client ARM64

Terms: If-Implemented

PWG-Raster

Printers supporting PWG Raster must conform to the PWG Raster industry standard and implement the following requirements.

1. Follow all the rules described in PWG 5102.4-2012.
2. Devices supporting duplex must support InsertPage directive.
3. Devices must consume all color formats and resolutions that are advertised in PrintDeviceCapabilities file.

4. 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.
5. Devices must support one of the following color formats
 - `Srgb_8`
 - `Cmyk_8`
 - `Sgray_8`
6. Devices supporting WS-Print v2.0 must indicate support for at least one `psk:PageResolution` which is less than or equal to 360dpi.
7. For mobile compatible mode, device must support 300 DPI output
8. 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.

1. 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

2. Device must support receiving content at 300 dpi
3. Device must support receiving content compressed using flate, run-length encoding or DCTDecode

Device.Imaging.Printer.USB

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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.

Device.Imaging.Printer.WSD

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 11 Client x64 Windows 11 Client ARM64

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>

Device.Imaging.Printer.XPS

Device.Imaging.Printer.XPS.XPS

Not applicable to v3 drivers and optional for v4 drivers

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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

Device.Imaging.Scanner.Base

Device.Imaging.Scanner.Base.dataTransfer

WIA drivers must support specific data transfer implementations.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Windows Image Acquisition (WIA) drivers must do the following:

- Use only the **Write**, **Seek**, and **SetSizeStream** methods during downloads.
- Use only the **Read**, **Seek**, and **StatStream** methods during uploads.
- Send valid **IPercentComplete** values during calls to the **IWiaMiniDrvTransferCallback::SendMessage**<element>. **IPercentComplete** 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 11 Client x64

Page **293** of **561**

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.msp> and "What's new in WIA 2.0" at [http://msdn.microsoft.com/en-us/library/ms630379\(VS.85\).aspx](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 11 Client x64
Windows 11 Client ARM64

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.

Device.Imaging.Scanner.WSD**Device.Imaging.Scanner.WSD.WSScan**

Scanners that have a network connection must implement the WS-Scan protocol.

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

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>.

Device.Input

Device.Input.Digitizer.Base

Device.Input.Digitizer.Base.NoiseSuppression

Digitizer reliability

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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).

Business Justification

This requirement ensures that only data corresponding to a user's input is being reported by the digitizer.

Device.Input.Digitizer.Base.HIDCompliant

HID compliant device firmware and/or HID mini-port driver

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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 [Windows Digitizer Class Input Devices | Microsoft Docs](#)

For HID over I2C requirements, see <http://msdn.microsoft.com/en-us/library/windows/hardware/Dn642101.aspx>

For HID over USB requirements, see [HID Usage Tables 1.22 | USB-IF](#)

For HID over Bluetooth requirements, see https://www.bluetooth.org/docman/handlers/downloadaddoc.ashx?doc_id=309012

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 11 Client x64 Windows 11 Client ARM64

Description

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, touchpad, 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 11 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.

Microsoft requires the touchpad module to support Microsoft Precision Touchpad via Firmware.

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.

Device.Input.Digitizer.Pen

Device.Input.Digitizer.Pen.Accuracy

Pen contact accuracy

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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:

- $\leq \pm 0.5$ mm (center)
- $\leq \pm 1$ 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 $\leq \pm 1$ 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.BarrelButton

Barrel Button Affordance

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

An integrated Windows pen solution shall have a hardware affordance on the pen that provides barrel button functionality compliant with the associated protocol. The barrel button functionality must be enabled by default.

Microsoft requires that all pens support barrel button functionality by default.

Business Justification

This requirement ensures that interactions with the pen are intuitive and efficient. This implementation enables users to have a consistent selection experience across all Windows applications using the pen.

Note: Active pens that enable barrel button functionality provide the most natural means to select digital ink, text, files, and other content. Active pens that enable barrel button functionality also free the application developer from having to develop UI to switch pen context from ink mode to selection mode. Therefore it is strongly recommended that all pens support barrel button functionality by default.

Device.Input.Digitizer.Pen.Buffering

Buffering for buses with High Resume latency

Applies to	Windows 11 Client 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 $\geq 100\text{ms}$ 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.CustomGestures

Custom Run-Time System Gestures

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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 requires that all Windows pen solution should 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 11 Client x64 Windows 11 Client ARM64

Description

See Device.Input.Digitizer.Base.HIDCompliant.

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.Pen.HighResolutionTimeStamp

Hardware Scan Time

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

One per frame, snapped in association to the frame sample time and not any other time in another stage of the pipeline for example, taking the time the scan started rather than when the packet is produced or transmitted. The time stamp can be taken at the beginning or end of the frame, but the setting should remain consistent. There is no need to synchronize it to any definition of absolute time. Use rollover for the time stamp, so there is no need to reset to zero. Timestamp should be 100 μ s units/resolution and be provided to the OS in a 16-bit (2 byte) field in the HID report. At any instance, allowable clock drift +/- 5% across standard operating temperatures (+25°C to + 85°C)

Business Justification

When a high-resolution time stamp is provided by the digitizer it can be leveraged by the OS and application frameworks for improved gesture recognition involving velocity and acceleration calculations.

Device.Input.Digitizer.Pen.HoverRange

Hover Range

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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.

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 11 Client x64 Windows 11 Client ARM64

Description

Reported jitter from an integrated Windows pen shall be:

- Moving Jitter
 - $\leq \pm 0.4$ mm (center)
 - $\leq \pm 0.6$ mm (within 3.5mm of all edges)
- Stationary Jitter
 - $\leq \pm 0.3$ mm (center)
 - $\leq \pm 0.5$ mm (within 3.5mm of all edges)
- Hover Stationary Jitter $\leq \pm 0.5$ mm (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.NoiseSuppression

Digitizer Reliability

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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).

Business Justification

This requirement ensures that only data corresponding to a user's input is being reported by the digitizer.

Device.Input.Digitizer.Pen.Pressure

Pressure Reporting

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 $\leq 20g$.

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.

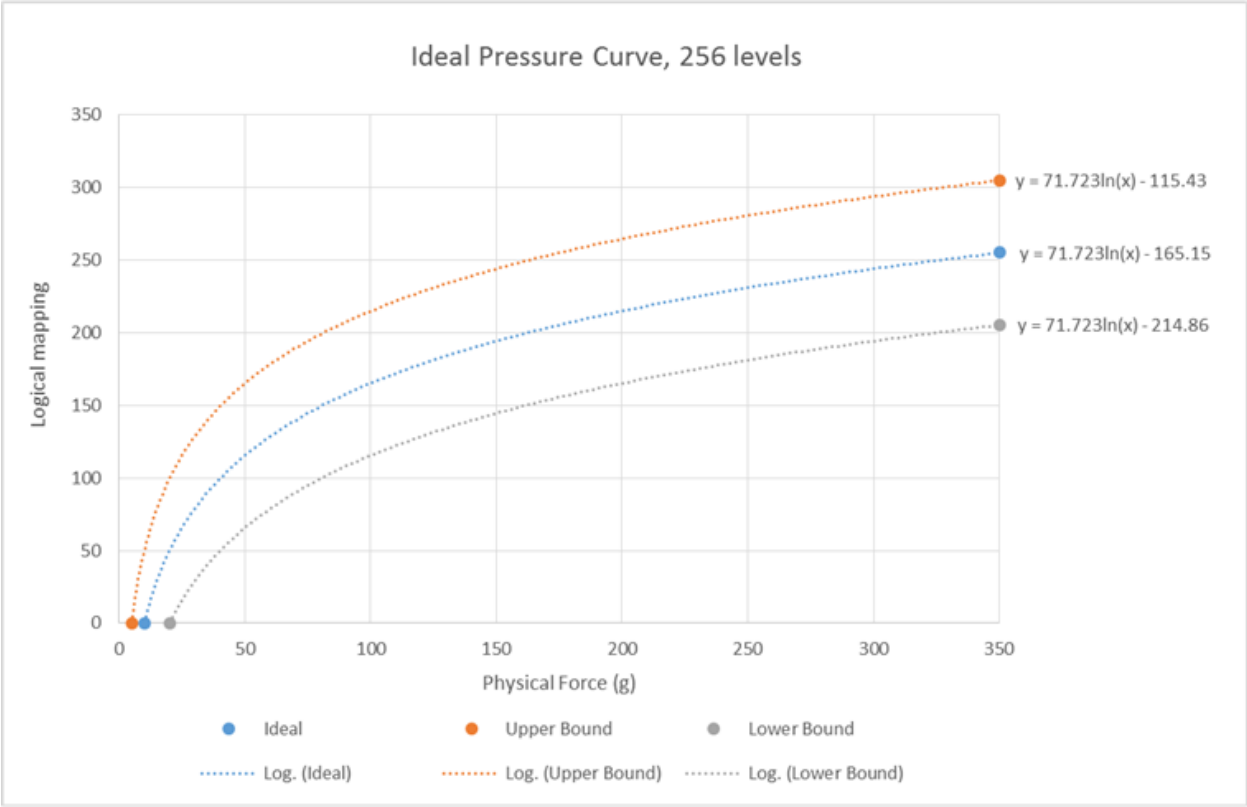


Figure 1 - Ideal Pressure Reporting Curve for a Device with 256 Pressure Levels

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 11 Client x64 Windows 11 Client ARM64

Description

An integrated Windows pen digitizer shall have a report rate $\geq 133\text{Hz}$.

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 11 Client x64 Windows 11 Client ARM64

Description

A pen digitizer's input resolution shall be $\geq$ 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.ResponseLatency

Response Latency

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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) $\leq 150\text{ms}$ (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) $\leq 35\text{ms}$ (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 $\leq 30\text{ms}$ (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.

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.ThirdPartyDrivers

Servicing and 3rd Party Driver Availability

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

See Device.Input.Digitizer.Base.ThirdPartyDrivers.

Business Justification

This requirement ensures that digitizers are always available for user input regardless of servicing scenarios and or clean installinds. There are instances where the digitizer may be the

Device.Input.Digitizer.PrecisionTouchpad

Device.Input.Digitizer.PrecisionTouchpad.Accuracy

Accuracy

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Windows precision touchpads shall report the absolute digitizer position ≤ 2 mm from actual position for all contacts.

Business Justification

Requirement ensures an accurate mousing and gesturing surface.

Device.Input.Digitizer.PrecisionTouchpad.Buffering

Buffering for buses with high resume latency

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

If Windows precision touchpads implement external buttons and are not click-pads 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.

Business Justification

Clickable regions should be comfortably usable. Windows 10 1607 changes increase clarity and reproducibility of the requirement. Previous manufacturing tolerances during testing are now built into the requirement.

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.ContactTipSwitchHeight

Contact Tip Switch Height

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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 11 Client 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](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 11 Client x64 Windows 11 Client ARM64

Description

Windows precision touchpads shall be ≥ 64 mm in length (horizontal measurement of touchpad) x ≥ 32 mm (vertical measurement). For touchpads that do not have a permanent physical indication of the active area (e.g. seamless), refer to the guidance in the Precision Touchpad Validation Guide. This guidance applies to all touchpad form factors, including but not limited to; seamless, extra buttons, dead zones, etc.

Exceptions to these dimensions may be granted under certain circumstances when a touchpad is positioned beside a keyboard (instead of above or below).

1. The minimum width of the touchpad is required to be 50mm. The maximum finger count will be 3 fingers if between 50-58mm, and 4 fingers if 58-64mm, otherwise 5 fingers are permissible.
2. The minimum surface of the touchpad should be at least 2048mm², but with the addendum that the minimum width must be 50mm.

Business Justification

Precision touchpads must maintain a minimum size to allow users to interact properly mousing while mousing across long distances and performing multi-finger gestures.

Device.Input.Digitizer.PrecisionTouchpad.FingerSeparation*Finger Separation*

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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 11 Client x64 Windows 11 Client ARM64

Description

See Device.Input.Digitizer.Base.HIDCompliant.

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.PrecisionTouchpad.HighResolutionTimeStamp

Hardware Scan Time

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

One timestamp per frame, snapped in association to the frame sample time and not any other time in another stage of the pipeline for example, taking the time the scan started rather than when the packet is produced or transmitted. The time stamp can be taken at the beginning or end of the frame, but the setting should remain consistent. There is no need to synchronize it to any definition of absolute time. Use rollover for the time stamp, so there is no need to reset to zero. Timestamp should be 100 μ s units/resolution and be provided to the OS in a 16-bit (2 byte) field in the HID report. At any instance, allowable clock drift +/- 5% across standard operating temperatures (+25°C to + 85°C).

Business Justification

When a high-resolution time stamp is provided by the digitizer it can be leveraged by the OS and application frameworks for improved gesture recognition involving velocity and acceleration calculations.

Device.Input.Digitizer.PrecisionTouchpad.InputResolution

Input Resolution

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

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 $\leq 2\text{mm}$ 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 mousing and gestures work correctly and that pausing during gesturing does not result in jittering visuals.

Device.Input.Digitizer.PrecisionTouchpad.MinMaxContacts

Contact count

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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. Multi-hand interactions are not suited for touchpad use, so more than 5 contacts support is unneeded.

Device.Input.Digitizer.PrecisionTouchpad.NoiseSuppression

Digitizer Reliability

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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).

Business Justification

This requirement ensures that only data corresponding to a user's input is being reported by the digitizer.

Device.Input.Digitizer.PrecisionTouchpad.ReportRate

Report Rates

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Windows precision touchpads shall report at $\geq 125\text{Hz}$ (and no greater than 250Hz) when a single contact is on the digitizer. Windows precision touchpads shall report all contacts at $\geq \text{Display Refresh Rate} + 10\text{Hz}$ (up to 125Hz) when multiple contacts are on the digitizer.

Business Justification

Maintain mouse experience requirements and competitive performance at $\geq 125\text{Hz}$. Gesture performance in-line with the smooth scrolling and zooming user experience seen on a mouse, which is further improved with high refresh rate displays.

Device.Input.Digitizer.PrecisionTouchpad.ResponseLatency

Response latency

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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.
 - Windows precision touchpads connected through Bluetooth shall have ≤ 55 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, mousing, 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.SelectiveReporting

Selective reporting

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

See Device.Input.Digitizer.Base.ThirdPartyDrivers.

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.

Device.Input.Digitizer.Touch

Device.Input.Digitizer.Touch.Accuracy

Accuracy

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

A touch digitizer shall report the position of all contacts within the following allowances:

- $\leq \pm 1\text{mm}$ when contact location is greater than 3.5mm from digitizer edge
- $\leq \pm 2\text{mm}$ 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 11 Client x64 Windows 11 Client ARM64

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 $\geq 100\text{ms}$ 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.CustomGestures

Custom run-time system gestures

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

Description

All contacts shall be uniquely tracked and reported when $\leq 8\text{mm}$ 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 11 Client x64 Windows 11 Client ARM64

Description

See Device.Input.Digitizer.Base.HIDCompliant.

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.Touch.HighResolutionTimeStamp

Hardware Scan Time

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

One per frame, snapped in association to the frame sample time and not any other time in another stage of the pipeline for example, taking the time the scan started rather than when the packet is produced or transmitted. The time stamp can be taken at the beginning or end of the frame, but the setting should remain consistent. There is no need to synchronize it to any definition of absolute time. Use rollover for the time stamp, so there is no need to reset to zero. Timestamp should be 100 μ s units/resolution and be provided to the OS in a 16-bit (2 byte) field in the HID report. At any instance, allowable clock drift +/- 5% across standard operating temperatures (+25°C to + 85°C).

Business Justification

When a high-resolution time stamp is provided by the digitizer it can be leveraged by the OS and application frameworks for improved gesture recognition involving velocity and acceleration calculations.

Device.Input.Digitizer.Touch.Jitter

Jitter and linearity

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Stationary Jitter shall not exceed a maximum of $\leq$ +/- 0.5mm away from the edges and $\leq$ +/- 1mm within 3.5mm of edges

Moving Jitter shall not exceed a maximum of $\leq$ +/- 1mm away from the edges and $\leq$ +/- 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.MinContactCount

Minimum simultaneous reportable contacts

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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.NoiseSuppression

Digitizer Reliability

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

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).

Business Justification

This requirement ensures that only data corresponding to a user's input is being reported by the digitizer.

Device.Input.Digitizer.Touch.ReportRate

Report rate

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

When in an active power state, the touch digitizer reporting rate for all contacts shall be maintained at a minimum of 60Hz and a maximum of 250Hz.

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 11 Client x64
Windows 11 Client ARM64

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.

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.ResponseLatency

Response latency

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

Description

A touch digitizer shall have response latencies as follows:

- Down Latency (Active) $\leq 35\text{ms}$
 - 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) $\leq 150\text{ms}$
 - 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
$< 7''$	$\leq 35\text{ms}$
$\geq 7''$	$\leq 25\text{ms}$

- 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.ThirdPartyDrivers

Servicing and 3rd Party Driver Availability

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

See Device.Input.Digitizer.Base.ThirdPartyDrivers.

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.

Device.Input.FingerPrintReader

Device.Input.FingerPrintReader.Base

Biometric fingerprint reader general requirements

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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](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.

All devices (peripheral or embedded) that leverage Windows Hello experiences for logon or user authentication must follow Windows Hello design language:

- Windows provided biometric enrollment applications must be used to collect the user's biometric data to generate a valid biometric template.
- Windows Setting page must be used to manage the enrolled biometric data.
- Windows feedback string above the lock screen (combined with glyphs) must be used to communicate to users about error cases (an unknown finger touched) or instructions (move away)

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	
Performance	* Fingerprint recognition speed: <350 ms from user touching sensor to recognition of the user for on host matching only. * Enrollment samples: <= 20 samples needed for enrollment * Enrollment success rate: Enrollment is successful 99% of the time	
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. USB devices should implement USB Selective Suspend.
Security	<u>Large area sensors (a sensor matrix of 160 x160 pixels or more at a dpi of 320 or greater)</u> * False Accept Rate (FAR): < 0.001% * Effective, real world FRR with Anti-spoofing or liveness detection: <5% * Presentation attack detection measures are a requirement. <u>Small area sensors (a sensor matrix of less than 160x160 at a dpi of 320 or greater)</u> * False Accept Rate (FAR): < 0.002% * Effective, real world FRR with Anti-spoofing or liveness detection: <5% * Presentation attack detection measures are a requirement. For those partners who desire more information on specific requirements behind the validation of presentation attack detection, please reach out to asreqs@microsoft.com.	
Sensor Type	Touch Sensor Only	Touch sensor type is the preferred option compared 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. Investments in presentation attack detection are also not mature for swipe sensors.
WBF Compliant with HCK	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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 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 11 Client x64 Windows 11 Client ARM64

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.
- 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.
Signature Requirements	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. Additionally, the driver INF must also report the WindowsHello signature attribute and list each adapter DLL (and any dependent DLLs) as receiving the attribute. For more information on INF signature attribute configuration for Windows Hello please see: https://msdn.microsoft.com/windows/hardware/drivers/install/inf-signatureattributes-section

Device.Input.FingerPrintReader.WBDI

Biometric fingerprint reader WBDI driver requirement

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 several 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>.

Device.Input.HID

Device.Input.HID.I2CProtocolSpecCompliant

All HID devices connected over I2C must comply with Microsoft HID I2C Protocol specification.

Applies to	Windows 11 Client 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 11 Client 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.

Device.Input.Keyboard

Logo requirements detailing the implementation details of a keyboard important to Microsoft operating systems

Device.Input.Keyboard.BrowserMultimediaKeysUseMSApis

Keys for Internet browser and multimedia must use Microsoft APIs.

Applies to	Windows 11 Client 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 glyphs.

Applies to	Windows 11 Client 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 11 Client 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 11 Client 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](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.msp> and <http://www.usb.org/developers/hidpage>.

Device.Input.Keyboard.KernelModeDriversUseWdfKmdf

Keyboard kernel-mode drivers must use the WDF-KMDF.

Applies to	Windows 11 Client 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 11 Client 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 11 Client 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 11 Client 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.

Device.Input.LampArray

These requirements are for LampArray devices interacting with the Dynamic Lighting System

Device.Input.LampArray.Accessibility

Special keys should be designed to contrast with the backlighting effect.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Established: 2308

Cost of Impacts: XS for OEMs/IHVs to implement

Enhancement Scenarios: This ensures that users will have good user experience on Dynamic Lighting devices.

Description

In the absence of a physical LED light to indicate the ON/OFF state of the following special keys, the device must highlight these keys by reporting a different color.

- Caps Lock
- Scroll Lock
- Num Lock
- Function keys

Scenario Example:

If a user selects the white effect for their keyboard, they may choose any color for the Caps or special keys mentioned above to stand out. If a user selects any lighting effect other than white for their keyboard, the special keys will default to white.

Business Justification

This requirement ensures consistent accessibility user experiences across dynamic lighting devices.

Device.Input.LampArray.AutonomousMode

The device should report a default lighting effect when in Autonomous Mode.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The OS controls Autonomous Mode. If nothing in the OS controls the device or if Dynamic Lighting is disabled, the device enters Autonomous Mode and reports its default lighting effect.

This requirement ensures that the device is reporting a default lighting effect so that when the device enters Autonomous Mode, users will still have a default lighting effect supported on the device.

Business Justification

This requirement ensures that when the OS is not controlling the lighting devices, the device will report a default lighting effect for the user.

Device.Input.LampArray.Brightness

Devices should demonstrate they can meet brightness requirements.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Dynamic Lighting compatible devices should meet minimum brightness requirements. This is a combined hardware affordance and firmware/software requirement. This will be measured "by eye" to start. When brightness is set to 0, there shouldn't be any light emitting from the device in a dark room.

Business Justification

To ensure a high quality and consistent user experience across devices, Dynamic Lighting-compatible devices must meet minimum LED brightness requirements.

Device.Input.LampArray.ColorAccuracy

Devices should accurately report and display color.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Dynamic Lighting compatible devices should share basic color definitions, reporting, and physical output. This is a combined hardware affordance and firmware/software requirement. Initially, these colors should match "by eye". I.e) A greenish yellow and a reddish yellow will both pass as yellow, as long as a reasonable person would say both colors are yellow. A good starting point is 255 color levels for R, G, B.

Business Justification

This requirement ensures consistent, expected user experiences with the colors on Dynamic Lighting devices. A Red color on a Fabrikam keyboard should be ~equivalent to Red on a Contoso CPU fan.

Device.Input.LampArray.ColorLevels

The device should report a color level of 255,255,255,255 for the selected lamp.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

By using the RGBBlue tool, you will be able to identify whether the firmware is correctly reporting a color level of 255,255,255,255 for each lamp.

Business Justification

This requirement ensures that each lamp accurately reports its color in the RGBI color model.

Device.Input.LampArray.Control

Devices should demonstrate they interact appropriately with the Dynamic Lighting System.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Dynamic Lighting compatible devices should enable control by and interact appropriately with the Dynamic Lighting System.

- Lighting control prioritization works correctly (notification, ambient, foreground)
- 3rd party ambient app should never override the lighting effect on Dynamic Lighting, unless the 3P app is higher in priority than the Dynamic Lighting Background Controller in Windows Settings
- Dynamic Lighting can override 3P Lighting apps on device when conflicts occur, unless Dynamic Lighting is turned off at user level

Business Justification

To ensure a high quality and consistent user experience across devices, Dynamic Lighting-compatible devices must interact appropriately with the Dynamic Lighting System.

Device.Input.LampArray.DeviceConnectionType

All devices should function the same whether they're connected via USB, Bluetooth, wireless with USB dongle, etc.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Dynamic-Lighting compatible devices should function the same, no matter how they're connected to the PC.

Business Justification

This requirement ensures consistent, expected user experiences with the same device.

Device.Input.LampArray.DeviceFriendlyName

The device must have an easily identifiable name so that the user can easily identify and configure it.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Each Dynamic Lighting device must have a user-friendly name to enable quick identification and configuration by the user.

For example, a device friendly name is "Fabrikam K47 Keyboard". The name should generally be the brand/manufacturer's name plus the model's retail name.

Business Justification

This requirement ensures that users will enjoy a seamless and effortless experience when interacting with their Dynamic Lighting devices.

Device.Input.LampArray.DynamicLightingAPIs

Devices should demonstrate that they correctly use the Dynamic Lighting APIs

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Dynamic Lighting compatible devices should correctly enumerate and identify themselves to the Dynamic Lighting APIs. The APIs should be able to individually address each Lamp on a device. . (ex. for keyboards, this means each key) Compliance with this requirement can be demonstrated with successful test results from the RGBBlue tool.

Business Justification

To ensure a high quality and consistent user experience across devices, Dynamic Lighting-compatible devices must identify and interact with the Dynamic Lighting APIs as designed.

Device.Input.LampArray.FWOverride

Device manufacturers can override LampArray firmware on specific keyboard LEDs.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

- For ease of adoption and differentiation, OEMs are only permitted to override LampArray Firmware on the following keyboard LEDs but not on any other keys or buttons that impact the Dynamic Lighting effect: Caps Lock
- Scroll Lock
- Num Lock
- LEDs with company/brand logo
- Function keys

Business Justification

This requirement enables consistent but differentiated experiences.

Device.Input.LampArray.HardwareID

Devices should have a unique hardware ID for consistent user experiences.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Dynamic-Lighting compatible devices should have a unique hardware ID so Windows can retain user settings on the device after device disconnection, PC restart, etc.

Scenario Example:

If a user plugs in the DUT to the same USB port, Windows will retain the L2 device settings for this device.

If a user plugs in the DUT to a different USB port, Windows will set the device settings to L1 global settings for this device.

Business Justification

This requirement ensures consistent, expected user experiences with the same device after disconnection, restarting, etc.

Device.Input.LampArray.HIDCompliant

Devices should demonstrate they follow the HID LampArray standard.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Dynamic Lighting compatible devices should use the HID LampArray standard accurately. Includes all mandatory static data in the HID LampArray spec. HID Validator test is passed.

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 lighting solution.

Device.Input.LampArray.LampArrayApp

RGB devices from an OEM/IHV must support LampArray if that OEM/IHV has a LampArray-compatible app.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Established: 2308

Cost of Impacts: S for OEMs/IHVs to implement

Enhancement Scenarios: New feature in OS. This enables continued durability of the HID LampArray open standard.

Description

The set of "background light controller" apps will become an allow list where each app is checked for compliance with this requirement.

Business Justification

If an OEM/IHV ships RGB devices and RGB apps, and their app can control LampArray devices, their own devices must support LampArray. Without this requirement, over time, each OEM/IHV will build proprietary non-LampArray protocols for their own devices while still supporting LampArray devices in their app. This will result in a re-fragmentation of the RGB HW market.

Device.Input.LampArray.LampArrayKind

Device should report the correct lamp array kind.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

LampArrayKind describes the type of physical device that contains the LampArray (e.g. keyboard/mouse/game controller etc.). This helps the Host know what Lamp Attributes it can expect and associate it with other HID devices (keyboard/mouse). The kind must use one of LampArrayKind* values from Lighting and Illumination spec in table 3.6.1.

Business Justification

Every LampArray is expected to have attributes describing the physical device that contains the LampArray.

Device.Input.LampArray.LampArrayPurpose

Device should report the correct lamp array purpose.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Established: 2308

Cost of Impacts: M/H for OEMs/IHV's to implement

Enhancement Scenarios: New feature in OS. For example: This impacts an app that looks for control devices versus presentation devices and does something different between them.

Description

LampPurposes identify the high-level purpose/s of the Lamp. This helps the Host determine what scenarios the Lamp can be used. The value must be composed of one or more LampPurposes* flags described in the table [3.6.2](#).

Business Justification

The proposed LampArray permits a much richer property model for both the LampArray and individual Lamps and finer control over the Lamp state.

Device.Input.LampArray.LampPositionTolerance

Device should accurately report the bounding boxes and key spacing within an offset tolerance of less or equal to 1cm.

Applies to	Windows 11 Client x64

Description

The purpose of the layout info is to enable apps to animate lighting across the device. Correct relative positions would be important to animations that rely on the positions of keys. For example: an effect that sweeps right to left on a display. If the key isn't where it's reported to be, it will light up at the wrong time.

Any errors in either relative positions or in absolute position would impact apps rendering the device. i.e. it might not look right in the app, and scale could be wrong.

Business Justification

This requirement ensures that the layout information is accurate, enabling apps to animate lighting across the device.

Device.Input.LampArray.LampID

Devices should report Lamp ID in sequence for consistent user experiences.

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

Established: 2308

Cost of Impacts: XS for OEMs/IHVs to implement

Enhancement Scenarios: New feature in OS. Enable partners to validate that their devices can accurately and smoothly play Dynamic Lighting effects in a consistent manner.

Description

Each lamp must have a unique Lamp ID that is assigned to lamps in sequence, starting from the top-left. You can identify the LED that is lit and extinguished in sequence in both forward and reverse directions by using the RGBBlue tool.

Business Justification

To ensure a high-quality and consistent user experience across devices, a Dynamic Lighting-compatible device must be capable of smoothly playing lighting effects with accurate direction on Windows.

Device.Input.LampArray.RefreshRate

Device should demonstrate they can meet refresh rate, latency, buffer requirements.

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

Description

All Dynamic Lighting compatible devices should meet minimum refresh rate and latency requirements. This is a combined hardware affordance and firmware/software requirement. All Lamps/LEDs on the device should meet the minimum requirement.

- All Dynamic Lighting compatible devices should have a report and refresh rate ≥ 30 Hz
 - HID descriptor MinUpdateIntervalInMicroseconds ≤ 33333.33 microseconds
- All Dynamic Lighting compatible devices should have response latency ≤ 5 ms

Business Justification

To ensure a high quality and consistent user experience across devices, Dynamic Lighting-compatible devices must meet minimum refresh rate and latency requirements.

Device.Input.LampArray.VirtualKeyMapping

Virtual keys should be reported to allow apps to find specific keys and light them up according to the input mapping.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The virtual key mapping is used to allow apps to find specific keys and light them up by the input mapping. For example, lighting up the "W" "A" "S" "D" keys require an input mapping.

Be sure that when RGBBlue tool says **VK_Q** that the **Q** key is showing up, etc...

Note: Fn keys do not emit any key events or character events in the OS. As such, there is no sensible input mapping. It would still be considered as passing the RGBBlue tool if you see those Fn keys reported as Unknown VKNone.

Business Justification

This requirement ensures that apps will be able to find specific keys and light them up by the input mapping.

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

Device.Input.Location.AssistedGNSS

Only relevant if the hardware uses the location stack for Assisted GNSS of the given type.

Applies to	Windows 11 Client x64

Terms: If-Implemented

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 11 Client x64

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
<code>SENSOR_DATA_TYPE_LATITUDE_DEGREES</code>	<code>VT_R8</code>	Shows the current latitude in degrees, which must be in the range [-90, +90]
<code>SENSOR_DATA_TYPE_LONGITUDE_DEGREES</code>	<code>VT_R8</code>	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

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 11 Client x64

Terms: If-Implemented

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 received.
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 received.
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

This requirement is applicable only to a GNSS driver integrating with the GNSS DDI introduced in Windows 10.

Applies to	Windows 11 Client x64

Terms: If-Implemented

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.V2UPL

This requirement is applicable only to a GNSS driver integrating with the GNSS DDI introduced in Windows 10.

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description

The V2 User Plane Location (V2UPL) requirement applies only to devices with CDMA cellular connections, and if required by the Mobile Operator.

Device.Input.PointDraw

Applies to mice, touch pads, and other input devices used to move the pointer

Device.Input.PointDraw.KernelModeDriversUseWdfKmdf

Mouse kernel-mode drivers must use the WDF-KMDF.

Applies to	Windows 11 Client x64

Description

Third-party mouse kernel-mode drivers must be ported to the WDF KMDF model.

Device.Input.Sensor

Device.Input.Sensor.Accelerometer

Device.Input.Sensor.Accelerometer.DataFieldProperties

Accelerometers must report the required data field properties

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Accelerometers must report the required data field properties. For more information, please refer to the [documentation](#).

Device.Input.Sensor.Accelerometer.DataTypes

Accelerometers are required to report acceleration data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Accelerometers are required to report acceleration data. For more information, see the [accelerometers data fields](#).

Device.Input.Sensor.Accelerometer.DynamicRange

Accelerometers are required to have a dynamic range of ± 2 G or more

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Accelerometers are required to have a sufficiently large dynamic range of ± 2 G or more.

Device.Input.Sensor.Accelerometer.MinimumReportInterval

Accelerometers are required to support a report interval of 40 milliseconds or less

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To provide a smooth and consistent accelerometer experience on Windows, accelerometers are required to support a report interval of 40 milliseconds or less. Supporting a report interval of 5 milliseconds or less is recommended.

Device.Input.Sensor.Accelerometer.Thresholds

Accelerometers are required to threshold on accelerometer data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Accelerometers are required to support thresholding on accelerometer force. For more information, see the [accelerometer thresholds](#).

Device.Input.Sensor.ActivitySensor

Device.Input.Sensor.ActivitySensor.DataTypes

Activity sensors are required to report the expected activity data.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Activity sensors are required to report activity type and activity confidence data. For more information, see [activity detection data fields](#).

Device.Input.Sensor.ActivitySensor.History

If an activity sensor supports history, then the sensor must support reporting a history of data samples

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Activity sensors are not required to support history. If an activity sensor does support history, then the history DDSI is required to be implemented along with reporting these properties:

Data Field	Data Type	Definition
PKEY_SensorHistory_MaxSize_Bytes	VT_UI4	The maximum size of sensor history data that the activity sensor can report in bytes.
PKEY_SensorHistory_Interval_Ms	VT_UI4	The sensor history sampling interval in milliseconds.
PKEY_SensorHistory_MaximumRecordSize_Bytes	VT_UI4	The maximum record size for history data.

Device.Input.Sensor.ActivitySensor.MinimumReportInterval

Activity sensors are required to support a report interval of 1 second or less.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Activity sensors are required to support a reporting interval of 1 second or less to provide consumers a consistent query of activity data.

Device.Input.Sensor.ActivitySensor.Thresholds

Activity sensors are required to support thresholding

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Activity Sensors are required to report a threshold value on the sensor's confidence percentage.

If the threshold parameter PKEY_SensorData_ActivityStream is set to TRUE, then the list of detected activities must not be filtered. The EvtSensorSetDataThresholds call must be blocked until any ongoing report to the class extension (SensorsCxSensorDataReady) is completed.

When PKEY_SensorData_ActivityStream is set to FALSE, the driver must filter the list of detected activities based on the subscribed activities defined in PKEY_SensorData_SubscribedActivityStates.

When thresholding, activity sensor must report a sample at the end of the report interval ONLY when any subscribed activity crosses the % confidence threshold (PKEY_SensorData_ConfidenceThreshold_Percentage) in either direction. This could be due to the confidence of subscribed activity changing or the class extension setting a different % confidence threshold.

The required threshold fields are shown below:

Data Field	Data Type	Definition
PKEY_SensorData_SubscribedActivityStates	VT_UI4	Indicates the activity states that should be reported. This is represented as a bitmask of ACTIVITY_STATE .
PKEY_SensorData_ActivityStream	VT_BOOL	Indicates if the activity sensor should stream. If this is true, no thresholding should be done. This means that all samples should be reported with all of the supported activity states.
PKEY_SensorData_ConfidenceThreshold_Percentage	VT_UI2	Indicates the confidence threshold.

Device.Input.Sensor.AmbientLightSensor

Device.Input.Sensor.AmbientLightSensor.AmbientLightResponseCurve

If a light sensor reports an ambient light response curve, then the reported curve must be well formed

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Ambient light sensors are not required to report an ambient light response curve. If an ambient light sensor does report an ambient light response curve, then it must follow the following:

Data Field	Data Type	Definition
PKEY_LightSensor_ResponseCurve	VT_VECTOR VT_UI4	The response curve of the sensor. This curve must have at least two points and the gradient must be positive or flat. For more information, see the response curve .

Device.Input.Sensor.AmbientLightSensor.ColorCapable

If a light sensor supports color detection, then the color related properties, thresholds, and data fields must be reported

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Ambient light sensors are not required to support color. If an ambient light sensor supports color then the following must be reported:

A color capable light sensor must report the following enumeration property:

Data Field	Data Type	Definition
DEVPKEY_LightSensor_ColorCapable	VT_BOOL	Specifies if this light sensor is color capable.

A color capable light sensor must report one of the following combinations of data fields:

- Lux, Kelvins, Chromaticity x, Chromaticity y
- Lux, Chromaticity x, Chromaticity y

For more information, see the [light sensor data fields](#).

For the color data fields that the light sensor reports, thresholding must also be supported. A sample must be reported when at least one threshold is met. For more information, see the [light sensor thresholds](#).

Device.Input.Sensor.AmbientLightSensor.DataFieldProperties

Ambient light sensors must report the required data field properties

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Ambient light sensors must report the required data field properties. For more information, please refer to the [documentation](#).

Device.Input.Sensor.AmbientLightSensor.DataTypes

Ambient light sensors are required to report lux data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Ambient light sensors are required to report light data. For more information, see [light sensor data fields](#).

Device.Input.Sensor.AmbientLightSensor.MinimumReportInterval

Ambient light sensors are required to support a sufficient minimum report interval

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Non-color capable ambient light sensors in Windows are required to support a report interval of 250 milliseconds or less. Color capable ambient light sensors in Windows are required to support a report interval of 1000 milliseconds or less.

Device.Input.Sensor.AmbientLightSensor.Thresholds

Ambient light sensors are required to support thresholding on lux data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Light sensors are required to support thresholding on lux.

If absolute threshold is supported, both the percent and absolute lux threshold must be met for a data sample to be reported. Some examples:

- Assume the absolute threshold is 1 lux and the percentage threshold is 25%:
 - If the last reported sample was 4 lux and the next sample is 3 lux:

- The next sample would get reported since the change is greater than or equal to 1 lux from the last reported sample *and* is greater than or equal to 25% of the last reported sample.
- If the last reported sample was 1 lux and the next sample is 0.5 lux:
 - The next sample would not get reported since the change is less than 1 lux from the last reported sample.
- If the last reported sample was 100 lux and the next sample is 90 lux:
 - The next sample would not get reported since the change is less than 25% of the last reported sample.

For more information, see [light sensor thresholds](#).

Device.Input.Sensor.Barometer

Device.Input.Sensor.Barometer.DataTypes

Barometers are required to report atmospheric pressure data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Barometers are required to report atmospheric pressure. For more information, see [barometer sensor data fields](#).

Device.Input.Sensor.Barometer.Thresholds

Barometers are required to support thresholding on atmospheric pressure data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Barometers are required to support thresholding on atmospheric pressure.

Data Field	Data Type	Definition
PKEY_SensorData_AtmosphericPressure_Bars	VT_R4	Specifies the absolute threshold for atmospheric pressure in bars.

Device.Input.Sensor.CustomSensor

Device.Input.Sensor.CustomSensor.DataTypes

Custom sensor are required to report the expected data fields

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Each data field in the custom sensor must be one of the supported data types, see [custom sensor data fields](#) for more information.

Device.Input.Sensor.Orientation

Device.Input.Sensor.Orientation.DataFieldProperties

Orientation sensors must report the required data field properties

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
Orientation sensors must report the required data field properties. For more information, please refer to the documentation .	

Device.Input.Sensor.Orientation.DataTypes

Orientation sensors are required to report orientation data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
Orientation sensors are required to report orientation data. For more information, see the orientation data fields .	

Device.Input.Sensor.Orientation.Thresholds

Orientation sensors are required to threshold orientation data correctly

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
Orientation sensors are required to support thresholding on this threshold key:	

Data Field	Data Type	Definition
PKEY_SensorData_RotationAngle_Degrees	VT_R4	Specifies the absolute threshold for the rotation angle in degrees

Device.Input.Sensor.Orientation.MinimumReportInterval

Orientation sensors are required to support a report interval of 16 milliseconds or less

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Orientation sensors are required to support a report interval of 16 milliseconds or less to provide a smooth orientation experience.

Device.Input.Sensor.General

Device.Input.Sensor.General.DataTypes

Sensors are required to report the expected data fields

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
All sensors required to report a time stamp. For more information, see the common sensor data fields .	

Device.Input.Sensor.General.DriverFramework

Sensors must no longer use the V1 sensor driver framework

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
The V1 sensor framework is deprecated. All location sensors must now be based on the V2 GNSS driver model, see this page for information. All other sensors must use the V2 sensors class extension, see this page for more information.	

Device.Input.Sensor.General.ReportInterval

Sensors are required to report data within the expected report interval

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
Over a period of time, a sensor must drop less than 1% of the expected number of samples if the report interval is 200 milliseconds or more. If the report interval is less than 200 milliseconds, a sensor must drop less than 5% of the expected number of samples.	

Device.Input.Sensor.General.StartStop

Sensors are required be able to start and stop

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
When a sensor starts, a data sample must be reported right within 1 second. When a sensor stops, no data should be reported.	

Device.Input.Sensor.GeoMagneticOrientation

Device.Input.Sensor.GeoMagneticOrientationSensor.DataTypes

Geomagnetic orientation sensors are required to report the expected data fields

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Geomagnetic orientation sensors are required to report orientation data. For more information, see [geomagnetic orientation data fields](#).

Device.Input.Sensor.GeoMagneticOrientationSensor.MinimumReportInterval

Geomagnetic orientation sensors are required to support a report interval of 16 milliseconds or less

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To provide a smooth and consistent geomagnetic orientation experience on Windows, geomagnetic orientation sensors are required to support a report interval of 16 milliseconds or less.

Device.Input.Sensor.GeoMagneticOrientationSensor.Thresholds

Orientation sensors are required to threshold correctly

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Geomagnetic orientation sensors are required to support thresholding on rotation angle.

Data Field	Data Type	Definition
PKEY_SensorData_RotationAngle_Degrees	VT_R4	Specifies the absolute threshold for the rotation angle in degrees

Device.Input.Sensor.GravityVectorSensor

Device.Input.Sensor.GravityVectorSensor.DataTypes

Gravity vector sensors are required to report a 3D vector indicating the direction and magnitude of gravity.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Gravity vector sensors are required to report acceleration due to gravity. For more information, see the [gravity vector data fields](#).

Device.Input.Sensor.GravityVectorSensor.Thresholds

Gravity vector sensors are required to support thresholding on a 3D vector indicating the direction and magnitude of gravity.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Gravity vectors are required to support thresholding on acceleration due to gravity.

Data Field	Data Type	Definition
PKEY_SensorData_AccelerationX_Gs	VT_R4	Specifies the acceleration force (due to gravity) threshold along the x-axis measured in units of the acceleration of gravity (G)
PKEY_SensorData_AccelerationY_Gs	VT_R4	Specifies the acceleration force (due to gravity) threshold along the y-axis measured in the units of the acceleration of gravity (G).
PKEY_SensorData_AccelerationZ_Gs	VT_R4	Specifies the acceleration force (due to gravity) threshold along the z-axis measured in the units of the acceleration of gravity (G).

Device.Input.Sensor.Gyrometer

Device.Input.Sensor.Gyrometer.DataFieldProperties

Gyrometers must report the required data field properties.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Gyrometers must report the required data field properties. For more information, please refer to the [documentation](#).

Device.Input.Sensor.Gyrometer.DataTypes

Gyrometers are required to report angular velocity data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Gyrometers are required to report angular velocity data. For more information, see the [gyrometer data fields](#).

Device.Input.Sensor.Gyrometer.MinimumReportInterval

Gyrometers are required to support a report interval of 16 milliseconds or less

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To provide a smooth and consistent gyrometer experience on Windows, gyrometers are required to support a report interval of 16 milliseconds or less. It is recommended to support a report interval of 5 milliseconds or less.

Device.Input.Sensor.Gyrometer.Thresholds

Gyrometers are required to support rate of rotation data thresholding.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Gyrometers are required to support thresholding on angular velocity. For more information, see [gyrometer thresholds](#).

Device.Input.Sensor.LinearAccelerometer

Device.Input.Sensor.LinearAccelerometer.DataTypes

Linear accelerometers are required to report acceleration data.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Linear accelerometers are required to report linear acceleration data. For more information, see the [linear accelerometer data fields](#).

Device.Input.Sensor.LinearAccelerometer.Thresholds

Linear accelerometers are required to support acceleration data thresholding.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Linear accelerometers are required to support thresholding on linear acceleration data.

Data Field	Data Type	Definition

PKEY_SensorData_AccelerationX_Gs	VT_R4	Specifies the acceleration force threshold along the x-axis measured, in units of the acceleration of gravity (G)
PKEY_SensorData_AccelerationY_Gs	VT_R4	Specifies the acceleration force threshold along the y-axis measured, in the units of the acceleration of gravity (G).
PKEY_SensorData_AccelerationZ_Gs	VT_R4	Specifies the acceleration force threshold along the z-axis measured, in the units of the acceleration of gravity (G).

Device.Input.Sensor.Magnetometer

Device.Input.Sensor.Magnetometer.DataFieldProperties

Magnetometers must report the required data field properties

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Magnetometers must report the required data field properties. For more information, please refer to the [documentation](#).

Device.Input.Sensor.Magnetometer.DataTypes

Magnetometers integrated into the system are required to expose all the required data fields

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Magnetometers are required to report magnetic field data. For more information see the [magnetometer data fields](#).

Device.Input.Sensor.Magnetometer.MinimumReportInterval

Magnetometers are required to support a report interval of 100 milliseconds or less

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To provide a smooth and consistent magnetometer experience on Windows, magnetometers are required to support a report interval of 100 milliseconds or less.

Device.Input.Sensor.Magnetometer.Thresholds

Magnetometers are required to support thresholding on magnetic field strength

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	Magnetometers are required to support thresholding on magnetic field strength. For more information, see magnetometer thresholds .

Device.Input.Sensor.Pedometer

Device.Input.Sensor.Pedometer.DataTypes

Pedometers are required to report step data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	Pedometers are required to report step type, step count, step duration, and if the pedometer has been reset. For more information, see pedometer data fields .

Device.Input.Sensor.Pedometer.History

Pedometers that implement history are required to support reporting a history of data samples

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	Pedometers are required to support history must implement all history DDSI functions and are required to report these history properties:

Data Field	Data Type	Definition
PKEY_SensorHistory_MaxSize_Bytes	VT_UI4	The maximum size of sensor history data that the pedometer can report in bytes.
PKEY_SensorHistory_Interval_Ms	VT_UI4	The sensor history sampling interval in milliseconds.
PKEY_SensorHistory_MaximumRecordSize_Bytes	VT_UI4	The maximum record size for history data.

Device.Input.Sensor.Pedometer.MinimumReportInterval

Pedometers are required to support a report interval of 1 second or less

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	

Pedometers are required to support a report interval of 1 second or less to provide consumers a consistent query of pedometer data.

Device.Input.Sensor.Pedometer.Thresholds

A pedometer are required to support thresholding based on cumulative pedometer step count

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Pedometers are required to support thresholding on the cumulative pedometer step count.

Data Field	Data Type	Definition
PKEY_SensorData_PedometerStepCount	VT_UI4	The number of steps detected

Device.Input.Sensor.ProximitySensor

Device.Input.Sensor.ProximitySensor.DataTypes

Proximity sensors are required to report the expected proximity data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Proximity sensors are required to report if an object is within proximity (PKEY_SensorData_ProximityDetection). For more information, see [proximity data fields](#).

Device.Input.Sensor.ProximitySensor.LongRangeProximity

If proximity distance is implemented by a proximity sensor, it is required to report the minimum and maximum range of detection

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Proximity sensors are not required to support proximity distance. If a proximity sensor does support reporting proximity distance, then the following requirements are required to be met:

The PKEY_SensorData_ProximityDistanceMillimeters data field must be reported and the sensor must report the following properties:

Data Field	Data Type	Definition
PKEY_SensorDataField_RangeMinimum	VT_R4	Indicates the lower boundary (inclusive) of the sensor's effective detection range in millimeters.
PKEY_SensorDataField_RangeMaximum	VT_R4	Indicates the upper boundary (inclusive) of the sensor's effective detection range in millimeters.

Device.Input.Sensor.ProximitySensor.ChangeNotification

Proximity sensors are required to only report data when the proximity detection data has changed

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

A driver implementing proximity is required to only report data when the underlying proximity data changes.

Device.Input.Sensor.HumanPresenceSensor

Description:

Proximity sensors are not required to support human presence. If a proximity sensor does support human presence, then the following requirements must be met:

Human Presence sensors must report if an object is within proximity (**PKEY_SensorData_ProximityDetection**). It must also report **PKEY_SensorData_ProximityDistanceMillimeters** data field and the the following properties:

Data Field	Data Type	Definition
PKEY_SensorDataField_RangeMinimum	VT_R4	Indicates the lower boundary (inclusive) of the sensor's effective detection range in millimeters.
PKEY_SensorDataField_RangeMaximum	VT_R4	Indicates the upper boundary (inclusive) of the sensor's effective detection range in millimeters.

Human Presence Sensors must report the enumeration property, **DEVPKEY_Sensor_ProximityType** to be **HumanProximity** which is part of the **ProximityType** enum.

Human Presence sensors are long range sensors and therefore must report a minimum detection distance of 1 meter. In other words, **PKEY_SensorDataField_RangeMinimum** should at least be 1 meter.

DEVPKEY_Sensor_ProximityType must report a value indicating proximity type is "Human" if the sensor supports human presence. **DEVPKEY_Sensor_WakeCapable** must be set to true if the human presence sensor is wake capable.

Device.Input.Sensor.HumanPresenceSensor.ApproachDetectionLatency

Human presence proximity sensors must timely report changes when a human approaches.

Applies to

Windows 11 Client x64

Description

Optional: If a proximity sensor supports human presence, then the sensor must meet the latency numbers defined below:

Proximity Sensor Type	Detection Latency
Camera Based	<1 second
Non-Camera Based	<0.25 seconds

Latency is defined as the time between the actual human approaching and the sensor reporting the change to the OS.

Device.Input.Sensor.HumanPresenceSensor.AttentionLatency

If Proximity Sensor supports human presence, and is capable of reporting attention then it should report attention within 500ms.

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

If Proximity Sensor supports human presence, and is capable of reporting attention then it should report attention within 500ms.

Device.Input.Sensor.HumanPresenceSensor.DetectionDistanceAccuracy

Human proximity sensors must be able to accurately report human presence within 5 cm.

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

If a proximity sensor implements human presence detection and supports reporting distance, then it must accurately be able to report a distance within 5 cm of where the actual human is.

Device.Input.Sensor.HumanPresenceSensor.DistanceReporting

Human presence proximity sensors that are not camera based must support distance reporting.

Applies to

Windows 11 Client x64

Description

If a proximity sensor implements human presence detection and is not camera based, then it must support reporting of distance. This means supporting the following data field:

- PKEY_SensorData_ProximityDistanceMillimeters

And the following thresholds:

- PKEY_SensorData_ProximityDistanceMillimeters
- PKEY_SensorData_HumanPresence_DetectionThreshold

Device.Input.Sensor.HumanPresenceSensor.DynamicRange

Human presence proximity sensors must support a minimum detection range.

Applies to**Windows 11 Client x64**

Description

If a proximity sensor implements human presence detection, then it must support a detection range of at least 0.1 meters to 1.2 meters. Though having a detection distance up to 3 meters is recommended.

Device.Input.Sensor.HumanPresenceSensor.ExtensionMechanismLatency

If Proximity Sensor supports human presence, and is leveraging the extension plug-in / broker process then the broker process should respond within 500ms.

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description

If Proximity Sensor supports human presence, and is leveraging the extension plug-in / broker process then the broker process should respond within 500ms.

Device.Input.Sensor.HumanPresenceSensor.FriendlyName

Human presence proximity sensors must report the DEVPKEY_NAME device interface property

Applies to**Windows 11 Client x64**

Description

If a proximity sensor implements human presence detection, then the sensor must report the DEVPKEY_NAME device interface property.

Device.Input.Sensor.HumanPresenceSensor.InboxProximitySensor

If a Proximity Sensor supports human presence, and then it must be exposed using the Inbox Proximity Sensor stack and not only a custom sensor stack.

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description

If a system has a proximity sensor that supports human presence, and then it must be exposed using the Inbox Proximity Sensor stack and not a custom sensor stack. It must set DEVPKEY_Sensor_ProximityType to HumanProximity.

Device.Input.Sensor.HumanPresenceSensor.IsValid

Human presence proximity sensors must report the "IsValid" data field.

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description

If a proximity sensor implements human presence detection, then the sensor must support the data field *PKEY_SensorData_IsValid*. This is useful in reporting conditions where the sensor may not be able to reliably determine human presence, such as if the sensor is occluded.

Device.Input.Sensor.HumanPresenceSensor.PowerConsumption

Human presence proximity sensors must be low power.

Applies to	Windows 11 Client x64

Description

If a proximity sensor supports human presence, then the power consumption should meet the numbers below:

Proximity Sensor Type	Modern Standby Power Consumption	Active Power Consumption
Camera Based	<25 mw	<80 mw
Non-Camera Based	<5 mw	<5 mw

This requirement is applicable to battery powered and AC powered systems.

Device.Input.Sensor.HumanPresenceSensor.USBSelectiveSuspend

Human presence proximity sensors that are USB based must support selective suspend.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

If a proximity sensor supports human presence and is USB based, then it must support selective suspend

Device.Input.Sensor.HumanPresenceSensor.ReportedProperties

If a Proximity Sensor supports human presence, and then it must be exposed using the Inbox Proximity Sensor stack and not only custom sensor stack.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

The following properties must be supported by the sensor:

PKEY_Sensor_Proximity_SensorCapabilities

DEVPKEY_Sensor_ProximityType

Device.Input.Sensor.RelativeOrientationSensor

Device.Input.Sensor.RelativeOrientationSensor.DataTypes

Relative orientation sensors are required to report the expected data fields

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Relative orientation sensors are required to report relative orientation data. For more information, see the [relative orientation sensor data fields](#).

Device.Input.Sensor.RelativeOrientationSensor.MinimumReportInterval

Relative orientation sensors are required to support a report interval of 16 milliseconds or less

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To provide a smooth and consistent relative orientation experience on Windows, relative orientation sensors are required to support a report interval of 16 milliseconds or less.

Device.Input.Sensor.RelativeOrientationSensor.Thresholds

Relative orientation sensors are required to support thresholding on rotation angle

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Relative orientation sensors are required to support thresholding on rotation angle.

Data Field	Data Type	Definition
PKEY_SensorData_RotationAngle_Degrees	VT_R4	Specifies the absolute threshold for the rotation angle in degrees.

Device.Input.Sensor.SimpleDeviceOrientation

Device.Input.Sensor.SimpleDeviceOrientation.DataTypes

Simple device orientation sensors are required to report simple device orientation data

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Simple orientation sensors are required to report the simple device orientation. For more information, see the [simple device orientation sensor data](#) fields.

Device.Input.Sensor.SimpleDeviceOrientation.MaximumReportDelay

Simple device orientation sensors are required to report a new orientation within 200 milliseconds of when an orientation change occurs or when the sensor starts

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
Simple device orientation sensors are required to report an orientation change within 200 milliseconds of when (1) an orientation change occurs or (2) the sensor starts. This allows consumers who rely on orientation changes to react to rotation changes in timely manner.	

Device.Input.Sensor.SimpleDeviceOrientation.SimpleDeviceOrientationChangeNotification

Simple device orientation sensors are required to only report data after the orientation data has changed

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
Description	
A driver implementing simple orientation sensor is required to only report data when the underlying sensor data changes. Except when the sensor is started, the driver must immediately compute the current device orientation and notify the class extension.	

Device.Input.SmartCardMiniDriver

MiniDriver program for smart cards

Device.Input.SmartCardMiniDriver.DoNotStopWhenResourcesAreUnavailable

Smart card driver must not stop the system if required resources are not available.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
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 Minidriver must meet Windows Smart Card Minidriver Version 5 Specifications and Certification Criteria.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64
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 logo requirements for each device category if a logo 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	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 v.Next	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.)

Device.Input.SmartCardMiniDriver.SupportMultipleInstancesOnASystem

Smart card driver can support multiple instances of the same device on a system.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

Device.Input.SmartCardReader

Device.Input.SmartCardReader.PinDataEntryKeyboardCompliesWithIso

An input device that implements a PIN data-entry keyboard must comply with ISO.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

Description

A smart card reader must support a minimum clock frequency of 3.5795 MHz.

Device.Input.SmartCardReader.SupportsMinDataRateOf9600bps

A smart card reader must support a minimum data rate of 9600 bps.

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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 11 Client 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 11 Client x64
Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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.

Device.Network.DevFund

Network requirements

Device.Network.DevFund.NdisVersion

NDIS devices must conform to the NDIS 6.x requirements in the Windows Driver Kit.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All NDIS device must conform to NDIS 6.x specified in the Windows Driver Kit. Please note that NDIS will continue to be supported for existing ethernet device classes as NetAdapterCx will begin supporting net new ethernet device class

Design Notes:

See the Windows Driver Kit, "NDIS."

Device.Network.DevFund.NPOS

Network Devices must support No Pause On Suspend (NPOS).

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.NetAdapterCxVersion

NetAdapterCx based devices must conform to the NetAdapterCx 2.2 in the Windows Driver Kit. Media Specific network adapters may have additional requirements, please reference specific Device requirements document.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

All net new NetAdapterCx based devices must conform to NetAdapterCx 2.2 version part of the Windows Driver Kit. This change allows advanced power management features for next generation ethernet devices shipping in modern consumer and enterprise form factors.

Design Notes:

See the Windows Driver Kit, "NetAdapterCx."

Device.Network.DevFund.SelectiveSuspend

NDIS devices must meet Selective Suspend requirements.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

NDIS devices must meet Selective Suspend requirements.

Device.Network.LAN.Base

LAN requirements

Device.Network.LAN.Base.100MbOrGreater

Ethernet devices must support 100-Mb or greater link speeds.

Applies to	Windows 11 Client x64

Description

Ethernet devices must be able to link at 100 Mb or higher speeds. Windows Server Ethernet devices must be able to link at 1Gbps or higher speeds.

Device.Network.LAN.Base.32MulticastAddresses

Ethernet devices must support filtering for at least 32 multicast addresses.

Applies to	Windows 11 Client 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 11 Client 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. . The latest modified version of the specification for the INF Keyword for Selective Suspend can be found at [Standardized INF Keywords for NDIS Selective Suspend](#). This includes a new keyword that specifies the idle time-out period and is only applicable when the screen is off.

Device.Network.LAN.Base.AnyBoundary

Ethernet devices must be able to transmit packets from buffers aligned on any boundary.

Applies to	Windows 11 Client 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 11 Client 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 11 Client 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". Please note that NDIS will continue to be supported for existing ethernet device classes as NetAdapterCx will begin supporting net new ethernet device class.

Device.Network.LAN.Base.NDISRequirements

Ethernet devices must conform to the NDIS requirements in the Windows Driver Kit.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All Ethernet device drivers must conform to NDIS as specified in the Windows Driver Kit. Design Notes: See the Windows Driver Kit, "NDIS." Please note that NDIS will continue to be supported for existing ethernet device classes as NetAdapterCx will begin supporting net new ethernet device.

Device.Network.LAN.Base.NetAdapterCxCalls

Net new ethernet devices must make only NetAdapterCx or WDF system calls.

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description

A driver for a net new ethernet device must make only NetAdapterCx or WDF calls. Any calls to other kernel mode components are not allowed. Design Notes: See the Windows Driver Kit, "NetAdapterCx" and "WDF."

Device.Network.LAN.Base.NetAdapterCxRequirements

All device drivers for net new ethernet devices must conform to the NetAdapterCx requirements in the Windows Driver Kit.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

All net new ethernet device drivers must conform to NetAdapterCx requirements as specified in the Windows Driver Kit. Design Notes: See the Windows Driver Kit, "NetAdapterCx"

Device.Network.LAN.Base.PacketFiltering

Ethernet devices must support packet filtering.

Applies to	Windows 11 Client 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 11 Client 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 11 Client 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 11 Client 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.SupportIEEE8023

Ethernet devices must comply with IEEE 802.3.

Applies to	Windows 11 Client x64

Description

All 802.3 Ethernet devices must implement and comply with the IEEE 802.3 specification.

Device.Network.LAN.ChecksumOffload

Network requirements

Device.Network.LAN.ChecksumOffload.ChecksumOffload

Ethernet devices must implement Checksum Offloads.

Applies to	Windows 11 Client 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.

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).

Device.Network.LAN.CS.NetAdapterCx

A driver for net new wired LAN (Ethernet) devices integrated into a Modern Standby system or docks shipped with the said system must be based on and conform to the NetAdapterCx 2.2 and above version and requirements in the Windows Drivers Kit (WDK).

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description: NetAdapterCx driver model support capabilities such as DFX, advanced wake scenarios and has inbuilt bugcheck resiliency meeting customer and OEM expectations for quality, reliability, and advanced feature support. The driver implementation must conform to the NetAdapterCx specification in the Windows Drivers Kit.

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client 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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64
Windows 11 Client ARM64

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.

Device.Network.LAN.DCB

LAN requirements

Device.Network.LAN.DCB.DCB

Ethernet devices that implement Data Center Bridging (DCB) must comply with the DCB Specification.

Applies to

Windows 11 Client 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.
MaxNumPfcEnabldTrafficClasses 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](http://msdn.microsoft.com/en-us/library/windows/hardware/hh451635(v=vs.85).aspx).

Device.Network.LAN.GRE

LAN requirements

Device.Network.LAN.GRE.GREPacketTaskOffloads

Ethernet devices that implement (NV) GRE Encapsulated Packet Task Offloads must comply with the specification.

Applies to	Windows 11 Client x64

Description

This requirement only applies to Ethernet devices that implement (NV) 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

Device.Network.LAN.IPsec

LAN requirements Note: IPsec task offload is deprecated but still supported in the product.

Device.Network.LAN.IPsec.IPsec

Ethernet devices that implement IPsec task offload must support required modes and protocols.

Applies to	Windows 11 Client 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\(v=WS.10\).aspx](http://technet.microsoft.com/en-us/library/dd125380(v=WS.10).aspx)
- Coexistence: Ethernet devices that support IPsec task offload and any the following offload technologies:
- Large Send Offload- Receive Side Scaling
 - CheckSum offload.

Devices implementing IPsec Task Offload and CheckSum offloads must implement them in a coexisting manner, such that the use of IPsec task offload does not preclude the use of CheckSum and other offload technologies implemented for each IPsec mode.

Device.Network.LAN.KRDMA

LAN requirements

Device.Network.LAN.KRDMA.KRDMA

Devices that implement the Network Direct 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 11 Client x64

Description

This requirement only applies to Ethernet devices that implement the Network Direct Kernel Mode Provider 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 2.0 has been the current specification since July 2014 and is now required (no submissions using Version 1.2 specification will be accepted). NDPKI Mode 3 (i.e. Guest RDMA) requirements are documented under Device.Network.LAN.SRIOV.VF.NDKMode3 and will only be required for Device.Network.LAN.SDDC.Premium submissions **after February 1, 2019** in the Guest VF Driver.

NDKPI version 2.0 describes three modes of operation:

- Mode 1, a mode that supports native mode operation;
- Mode 2, a mode that supports NDKPI on NIC_Switch vPorts; and
- Mode 3, a mode that supports NDKPI exposed on a VF driver.

Windows v10.0 Client drivers are only required to support Mode 1 and Mode 2.

Windows Server 2019 drivers must support Modes 1 and 2.

Windows Server 2019 and later with VF drivers that also support SR-IOV should support NDK Mode 3. This will be required for WSSD Premium and Azure Stack solutions after February 1, 2019.

Design Notes:

See the Network Direct Kernel Mode Interface (NDKPI) Specification (version 2.0).

Device.Network.LAN.LargeSendOffload

Network requirements

Device.Network.LAN.LargeSendOffload.LargeSendOffload

Ethernet devices must implement large send offloads.

Applies to	Windows 11 Client 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."

Device.Network.LAN.MSI

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.

In addition, the device must allocate at least 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.MTUSize

Device.Network.LAN.MTUSize

Applies to	Windows 11 Client 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.

Device.Network.LAN.MTUSize.EncapOverhead

Applies to	Windows 11 Client x64

Description

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.

Devices implementing GRE or VxLAN offloads must also incorporate this requirement.

Device.Network.LAN.NDISPollMode

Device.Network.LAN.NDISPollMode

If implemented, NDIS 6.86 NIC drivers and later must support the NDIS Poll Mode execution model as defined by the NDIS Poll Mode execution model specification and demonstrate support by passing the relevant HLK tests.

Applies to	Windows 11 Client x64

Description

Drivers implementing NDIS 6.86 operating in poll-mode enable an improvement over the NDIS push-mode which is interrupt-based and susceptible to long-running Deferred Procedural (DPC) chains as a result of high inbound packet rates. This leads to system instability and/or starvation of an application's network traffic.

This requirement applies to high-speed adapters (10 Gbps and higher).

In a future release, NDIS Poll Mode will be mandatory.

Device.Network.LAN.PM

LAN requirements

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 11 Client x64 Windows 11 Client ARM64

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 11 Client 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 11 Client 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.

Device.Network.LAN.QoSOffload

LAN requirements

Device.Network.LAN.QosOffload.QosOffload

Ethernet devices that implement Hardware Qos Offload (HWQoS) must comply with the HW Qos Specification.

Applies to	Windows 11 Client x64

Description

This requirement applies to Ethernet devices that implement Hardware Qos Offload. Ethernet devices that implement Hardware Qos Offloads must comply with the specification and support the Max Cap per vPort Qos offload.

Device.Network.LAN.RSC

LAN requirements

Device.Network.LAN.RSC.RSC

Ethernet devices that implement Receive Segment Coalescing (RSC) must comply with the RSC Specification.

Applies to	Windows 11 Client x64

Description

Ethernet devices that implement Receive Segment Coalescing (RSC) must comply with the RSC Specification and require both IPv4 and IPv6 support. If RSC is implemented as a software offload, please disable in the miniport driver as the feature has been implemented in the virtualization host.

Design Notes:

NDIS version: 6.30

Device.Network.LAN.RSS

LAN requirements

Device.Network.LAN.RSS.RSS

Ethernet devices that implement RSS

Applies to	Windows 11 Client 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, UDP IPv4, IPv6, TCP IPv6, and UDP IPv6
See <https://docs.microsoft.com/en-us/windows-hardware/drivers/network/rss-hashing-types> It is mandatory to support 4-tuple UDP RSS by default and fall back to two-tuple RSS when the UDP header is not present (for e.g. in IP fragments). It is required that NDIS_RECEIVE_SCALE_CAPABILITIES advertisement include the new hash types NDIS_RSS_CAPS_HASH_TYPE_UDP_IPV4 and NDIS_RSS_CAPS_HASH_TYPE_UDP_IPV6 and optionally NDIS_RSS_CAPS_HASH_TYPE_UDP_IPV6_EX.
- Multiple processor groups (for miniports of NDIS version 6.20 and above)
- Minimum number of queues (depending on link speed):
 - For 1G NICs: 2 queues
 - For 10G+ NICs: 8 queues
 - For 40G+ NICs: 16 queues
 - 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
- Minimum number of RSS processors supported on the physical NIC (i.e. aggregate of all vPorts) is:

- For 1G NICs: 2 RSS processors
- For 10G+ NICs: 8 RSS processors
- For 40G+ NICs: 16 RSS processors

- 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

Design Notes:

See RSS Standardized Keywords Specification

[http://msdn.microsoft.com/library/windows/hardware/ff570864\(v=vs.85\).aspx](http://msdn.microsoft.com/library/windows/hardware/ff570864(v=vs.85).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 11 Client x64

Description

This requirement only applies to Ethernet devices that implement RSS. 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](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 11 Client x64

Description

This requirement only applies to Ethernet devices that implement RSS. 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, NumberOfLsb variable. An RSS implementation must support the host protocol stack setting NumberOfLsb 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 11 Client x64

Description

This requirement only applies to Ethernet devices that implement RSS. 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](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 11 Client x64

Description

This requirement only applies to Ethernet devices that implement RSS. At any time a network device supports RSS, it must support setting OID_GEN_RECEIVE_SCALE_PARAMETERS, including updating the Indirection Table, NumberOfLsb, 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 NumberOfLsb 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."

Device.Network.LAN.SDDC.Premium

Defines requirements that must be met if the LAN card (NIC) is supported in a WSSD Premium solution

Device.Network.LAN.SDDC.Premium.BasicFunction

Basic requirements for LAN cards used in Microsoft Windows Server Software Defined Premium Solutions

Applies to	Windows 11 Client x64

Description

Microsoft WSSD Premium solution requirements for LAN cards (NICs) are captured by the following table.

Feature	Requirement
Device.Network.LAN.10GbOrGreater	Device.Network.LAN.10GbOrGreater.CloudStress
Device.Network.LAN.VXLAN	Device.Network.LAN.VXLAN.VXLANPacketTaskOffloads
Device.Network.LAN.GRE	Device.Network.LAN.GRE.GREPacketTaskOffloads
Device.Network.LAN.QosOffload	Device.Network.LAN.QosOffload.QosOffload
Device.Network.LAN.Timestamp	Device.Network.LAN.Timestamp.Timestamp
Device.Network.LAN.USO	Device.Network.LAN.USO.USO
Device.Network.LAN.VMQ	Device.Network.LAN.VMQ.VirtualMachineQueues
Device.Network.LAN.VMMQ	Device.Network.LAN.VMMQ.VirtualMachineMultipleQueues
	Device.Network.LAN.VMMQ.RSSv2
Device.Network.LAN.RSS	Device.Network.LAN.RSS.RSS
	Device.Network.LAN.RSS.SetHashFunctionTypeAndValue
	Device.Network.LAN.RSS.SupportIndirectionTablesSizes
	Device.Network.LAN.RSS.SupportToeplitzHashFunction
	Device.Network.LAN.RSS.SupportUpdatesToRSSInfo
Device.Network.LAN.MTUSize	Device.Network.LAN.MTUSize
	Device.Network.LAN.MTUSize.EncapOverhead
Device.Network.LAN.LargeSendOffload	Device.Network.LAN.LargeSendOffload.LargeSendOffload
Device.Network.LAN.KRDMA	Device.Network.LAN.KRDMA.KRDMA
	Device.Network.LAN.KRDMA.KRDMA Mode 3 (Guest RDMA)
Device.Network.LAN.SRIOV	Device.Network.LAN.SRIOV.SRIOV
Device.Network.LAN.SRIOV.VF	Device.Network.LAN.SRIOV.VF.VF
	Device.Network.LAN.SRIOV.VF.NDKMode3
Device.Network.LAN.DCB	Device.Network.LAN.DCB.DCB
Device.Network.LAN.ChecksumOffload	Device.Network.LAN.ChecksumOffload.ChecksumOffload
Device.Network.LAN.Base	Device.Network.LAN.Base.100MbOrGreater
	Device.Network.LAN.Base.32MulticastAddresses
	Device.Network.LAN.Base.AdvProperties
	Device.Network.LAN.Base.AnyBoundary
	Device.Network.LAN.Base.IPv4AndIPv6OffloadParity

Feature	Requirement
	Device.Network.LAN.Base.NDISCalls
	Device.Network.LAN.Base.NDISRequirements
	Device.Network.LAN.Base.PacketFiltering
	Device.Network.LAN.Base.PreserveOSServices
	Device.Network.LAN.Base.PriorityVLAN
	Device.Network.LAN.Base.ShortPacketPadding
	Device.Network.LAN.Base.SupportIEEE8023

Device.Network.LAN.SDDC.Premium.CloudStress

Network controllers that are used for Microsoft Windows Server Software-Defined (WSSD) Premium features must comply with this specification

Applies to	Windows 11 Client x64

Description

Private cloud solutions comprise of tightly integrated software and hardware components to deliver resiliency with high performance. Issues in any of the components (software, hardware, drivers, firmware, and so forth) can compromise the solution and undermine any promises made regarding the Service Level Agreement (SLA) for the private cloud.

Many of these issues are surfaced only under a high-stress, private cloud simulation. The Private Cloud Simulator (PCS) enables you to validate your component in a cloud scenario and identify such issues. It simulates a live datacenter/private cloud by creating VM workloads, scheduling administrative operations (load balancing, software/hardware maintenance), and injecting faults (unplanned hardware/software failures) on the private cloud.

To comply with this specification, the device must pass the PrivateCloudSimulator - Device.Network.LAN.SDDC.Premium HLK test on a 4-node clustered configuration.

Device.Network.LAN.SDDC.Standard

Defines requirements that must be met if the LAN card (NIC) is supported in a Microsoft WSSD Standard solution

Device.Network.LAN.SDDC.Standard.Functions

Basic requirements for LAN cards used in Microsoft WSSD Standard offering

Note: Some features have been added to SDDC.Standard requirements (previously premium) leading up to the next LTSC release.

Applies to	Windows 11 Client x64

Description

Microsoft WSSD requirements for LAN cards (NICs) are captured by the following table.

Note: Some features have been added to SDDC.Standard requirements (previously premium) leading up to LTSC vNext.

Feature	Requirement
Device.Network.LAN.10GbOrGreater	Device.Network.LAN.10GbOrGreater.CloudStress
Device.Network.LAN.VXLAN	Device.Network.LAN.VXLAN.VXLANPacketTaskOffloads
Device.Network.LAN.VMQ	Device.Network.LAN.VMQ.VirtualMachineQueues
Device.Network.LAN.VMMQ	Device.Network.LAN.VMMQ.VirtualMachineMultipleQueues
	Device.Network.LAN.VMMQ.RSSv2
Device.Network.LAN. RSS	Device.Network.LAN.RSS.RSS
	Device.Network.LAN.RSS.SetHashFunctionTypeAndValue
	Device.Network.LAN.RSS.SupportIndirectionTablesSizes
	Device.Network.LAN.RSS.SupportToeplitzHashFunction
	Device.Network.LAN.RSS.SupportUpdatesToRSSInfo
Device.Network.LAN.MTUSize	Device.Network.LAN.MTUSize
	Device.Network.LAN.MTUSize.EncapOverhead
Device.Network.LAN.LargeSendOffload	Device.Network.LAN.LargeSendOffload.LargeSendOffload
Device.Network.LAN.KRDMA	Device.Network.LAN.KRDMA.KRDMA
	Device.Network.LAN.KRDMA.KRDMA Mode 3 (Guest RDMA)
Device.Network.LAN.SRIOV	Device.Network.LAN.SRIOV.SRIOV
Device.Network.LAN.SRIOV.VF	Device.Network.LAN.SRIOV.VF.VF
	Device.Network.LAN.SRIOV.VF.NDKMode3
Device.Network.LAN.GRE	Device.Network.LAN.GRE.GREPacketTaskOffloads
Device.Network.LAN.DCB	Device.Network.LAN.DCB.DCB
Device.Network.LAN.ChecksumOffload	Device.Network.LAN.ChecksumOffload.ChecksumOffload
Device.Network.LAN.Base	Device.Network.LAN.Base.100MbOrGreater
	Device.Network.LAN.Base.32MulticastAddresses
	Device.Network.LAN.Base.AdvProperties
	Device.Network.LAN.Base.AnyBoundary
	Device.Network.LAN.Base.IPv4AndIPv6OffloadParity
	Device.Network.LAN.Base.NDISCalls
	Device.Network.LAN.Base.NDISRequirements
	Device.Network.LAN.Base.PacketFiltering
	Device.Network.LAN.Base.PreserveOSServices

Feature	Requirement
	Device.Network.LAN.Base.PriorityVLAN
	Device.Network.LAN.Base.ShortPacketPadding
	Device.Network.LAN.Base.SupportIEEE8023

Device.Network.LAN.SDDC.Standard.CloudStress

Network controllers that are used for Microsoft SDDC offering must comply with this specification

Applies to	Windows 11 Client x64

Description

SDDC solutions comprise of tightly integrated software and hardware components to deliver resiliency with high performance. Issues in any of the components (software, hardware, drivers, firmware, and so forth) can compromise the solution and undermine any promises made regarding the Service Level Agreement (SLA) for the private cloud.

Many of these issues are surfaced only under a high-stress, private cloud simulation. The Private Cloud Simulator (PCS) enables you to validate your component in a cloud scenario and identify such issues. It simulates a live datacenter/private cloud by creating VM workloads, scheduling administrative operations (load balancing, software/hardware maintenance), and injecting faults (unplanned hardware/software failures) on the private cloud.

To comply with this specification, the controller must pass the PCS test run with the 'SDN' profile on a 4-node clustered configuration.

Device.Network.LAN.SRIOV

Network requirements

Device.Network.LAN.SRIOV.SRIOV

Ethernet devices that implement Single Root I/O Virtualization (SR-IOV) must support required functionalities.

Applies to	Windows 11 Client x64

Description

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_RECEIVED_FILTER_CAPABILITIES.

An 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 pairs not allocated to IOV must 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.

Device.Network.LAN.SRIOV.VF

Network requirements for Guest VF Category

Device.Network.LAN.SRIOV.VF.VF

Ethernet devices that implement Single Root I/O Virtualization (SR-IOV) must support required functionalities in the guest VM connected to SR-IOV enabled adapter.

Applies to	Windows 11 Client x64

Description

Requirements for an SRIOV VF adapter

If RSS is supported for VF miniports, the NIC must be able to support the same number of RSS queues as it can on PFs.

All NDIS devices must conform to NDIS 6.x specified in the Windows Driver Kit. VF devices must comply with the same level of NDIS 6.x support.

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 must support both IPv4 and IPv6.

On report of media connected, the VF miniport must be ready to accept traffic.

The implementation must comply with the Single Root I/O Virtualization Reference Specification.

Design Notes:

See the Single Root I/O Virtualization Specification.

Device.Network.LAN.SRIOV.VF.NDKMode3

Requirements for Ethernet VF devices that implement NDK Mode 3 (Guest RDMA) over SR-IOV. Required for WSSD Premium certified systems after February 1, 2019.

Applies to	Windows 11 Client x64

Description

NDPKI Mode 3 (i.e. Guest RDMA) requirements will only be required for Device.Network.LAN.SDDC.Premium and Device.Network.LAN.AzureStack submissions **after February 1, 2019**. This feature must be supported for each fabric (RoCE and/or iWarp) supported by the physical NIC.

Device.Network.LAN.Timestamp

LAN requirements

Device.Network.LAN.Timestamp.Timestamp

Ethernet devices that implement Hardware Timestamping (HWTimestamping) must comply with the HW Timestamping Specification.

Applies to	Windows 11 Client x64

Description

This requirement applies to Ethernet devices that implement Hardware Timestamping. Ethernet devices that implement Hardware Timestamping must comply with the specification.

Device.Network.LAN.UDP

Defines requirements for Ethernet devices that implement UDP Segmentation Offload

Device.Network.LAN.UDP.SegmentationOffload

Ethernet devices that implement UDP Segmentation Offload must comply with the UDP Segmentation Offload Reference specification.

Applies to	Windows 11 Client x64

Description

UDP Segmentation Offload (USO) reduces CPU utilization in UDP based protocols such as QUIC. It does this by offloading the segmentation of UDP datagrams that are larger than the MTU of the network. The requirements for USO closely resemble the existing requirements for LSOv2 for TCP. The implementation must comply with the UDP Segmentation Offload Specification

Device.Network.LAN.VMMQ

Defines requirements for Ethernet devices that implement Virtual Machine Multiple Queues

Device.Network.LAN.VMMQ.VirtualMachineMultipleQueues

Requirements for Ethernet devices that implement Virtual Machine Multiple Queues

Applies to	Windows 11 Client x64

Description

The implementation must comply with the Virtual Machine Multiple Queues Reference Specification. If implemented, initial value for Virtual Machine Multiple Queues must be Enabled.

Device.Network.LAN.VMMQ.RSSv2

Requirements for Ethernet devices that implement RSSv2

Applies to

Windows 11 Client x64

Description

The implementation must comply with the RSSv2 Reference Specification. This capability enables Dynamic VMMQ for automatic tuning and coalescing of queues per vPort.

Device.Network.LAN.VMQ

LAN requirements

Device.Network.LAN.VMQ.VirtualMachineQueues

Ethernet devices that implement Virtual Machine Queues must comply with the Programmable Machine Queues Reference specification.

Applies to

Windows 11 Client x64

Description

The implementation must comply with the Programmable Machine Queues Reference Specification.

At least thirty-two queues with filters must be supported. Support for at least 64 queues with filters is recommended.

The number of queues required will be 64 by December 1, 2019 for devices operating at speeds above 10

Gigabits/second. 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
32-64	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 at least 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 at least equal to the Number of queues

Total MAC addresses (NumTotalMacAddresses) must be at least equal to the Number of queues

MAC addresses per VM Queue (NumMacAddressesPerVMQueue) must be at least 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](http://msdn.microsoft.com/en-us/library/windows/hardware/ff571034(v=vs.85).aspx)

Device.Network.LAN.VXLAN

Device.Network.LAN.VXLAN.VXLANPacketTaskOffloads

Applies to	Windows 11 Client 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

Device.Network.MobileBroadband.CDMA(All CDMA test cases are optional – they are not required for certification)

Mobile broadband

Device.Network.MobileBroadband.CDMA.Discretionary

Mobile broadband devices that work with CDMA cellular class should support exchange interface based on the Mobile Broadband Driver Model.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: Optional

Description

CDMA 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](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/netpmcspc.rtf>

CDMA Mobile Broadband devices could support more features which are defined in previous Windows release and most of them will work as it is. But those features are no longer supported and not guaranteed working on the latest Windows release.

Device.Network.MobileBroadband.ComplyWithBaseReq

Mobile broadband devices must comply with the following base requirements.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 comply with both hardware and software radio 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](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/netpmmspc.rtf>

There are multiple tests that come under the base-requirement. They are listed below:

- 1) MBBDriverStress : The purpose of this test case is to stress test the class-driver. Hence the test creates multiple threads, and from each thread, invoke some OID operations such as query deviceCaps, query radiostate, connect/disconnect etc. The test validates that all the operations are executed successfully.

References :

Multiple OID are being queried/executed, a concise list of all OIDs can be found here : Network Driver Design Guide - Windows drivers | Microsoft Learn – section “Kernel Mode SDK topics for Network Drivers – Ntddndis.h”

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator.

Test parameters :

AccessString : Specifies the access string to use when making a connection

UserName : Specifies the user name to use when making a connection (optional)

Password : Specifies the password to use when making a connection (optional) WebDoc8K : Specifies a 8K Web Doc to download (optional)

DevPhoneNumber : Specifies the device's phone number

DevPhoneNumberType : NATIONAL, INTERNATIONAL, or UNKNOWN

- 2) TestSignalLoss : This test has 2 sub-tests which validate the following:
 - a. The test verifies the device provides status indications in the right sequence when signal strength loss occurs when packet service was attached.
 - b. The test verifies the device provides status indications in the right sequence when signal strength loss occurs when context was activated.

References:

OID_WWAN_SIGNAL_STATE - Windows drivers | Microsoft Learn

OID_WWAN_PACKET_SERVICE - Windows drivers | Microsoft Learn

OID_WWAN_REGISTER_STATE - Windows drivers | Microsoft Learn

OID_WWAN_CONNECT - Windows drivers | Microsoft Learn

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator. The tester should also have a facility to completely shutoff cellular signal (either have access to an area without cellular signal from the provider or use a Faradays cage)

Test parameters :

AccessString : Specifies the access string to use when making a connection

UserName : Specifies the user name to use when making a connection (optional)

Password : Specifies the password to use when making a connection (optional)

- 3) TestSimBad : The test verifies proper behavior when a bad SIM is inserted.

References:

[OID_WWAN_READY_INFO - Windows drivers | Microsoft Learn](#)

[OID_WWAN_DRIVER_CAPS - Windows drivers | Microsoft Learn](#)

[OID_WWAN_DEVICE_CAPS_EX - Windows drivers | Microsoft Learn](#)

[OID_WWAN_RADIO_STATE - Windows drivers | Microsoft Learn](#)

[OID_WWAN_SIGNAL_STATE - Windows drivers | Microsoft Learn](#)

[OID_WWAN_PIN - Windows drivers | Microsoft Learn](#)

[OID_WWAN_PIN_EX - Windows drivers | Microsoft Learn](#)

[OID_WWAN_PIN_LIST - Windows drivers | Microsoft Learn](#)

[OID_WWAN_HOME_PROVIDER - Windows drivers | Microsoft Learn](#)

[OID_WWAN_SMS_CONFIGURATION - Windows drivers | Microsoft Learn](#)

[OID_WWAN_SMS_STATUS - Windows drivers | Microsoft Learn](#)

[OID_WWAN_VISIBLE_PROVIDERS - Windows drivers | Microsoft Learn](#)

[OID_WWAN_CONNECT - Windows drivers | Microsoft Learn](#)

[OID_WWAN_PACKET_SERVICE - Windows drivers | Microsoft Learn](#)

[OID_WWAN_REGISTER_STATE - Windows drivers | Microsoft Learn](#)

[OID_WWAN_PREFERRED_PROVIDERS - Windows drivers | Microsoft Learn](#)

[OID_WWAN_PROVISIONED_CONTEXTS - Windows drivers | Microsoft Learn](#)

Test setup: This test requires a Mobile-Broadband supported test device and a corrupted (BAD) SIM card. Any SIM card that can trigger a readyState notification of MbbReadyStateBadSim will work. Tester can choose to edit a SIM card and modify its files to corrupt it (modem vendors/IHVs have specific tools to aid this process)

Test parameters:

None

Device.Network.MobileBroadband.NRNSA

Mobile broadband

Device.Network.MobileBroadband.NRNSA.Discretionary

Mobile broadband devices that implement New Radio Non-Standalone (NRNSA) must support exchange interface based on the Mobile Broadband Driver Model.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

Windows Mobile Broadband Driver Model is updated to support 5G radio technology network option 3. This is called NR NSA. Mobile broadband devices that support this feature must support RSRP/SNR reporting in signal strength indications. Additionally, NR frequency range information shall be supported.

NRNSA includes the following HLK test case :

1. Test5GNSA

Description : This HLK test verifies whether the device supports 5GNSA capability and if supported, it does some basic validations as explained further. The test first forces the modem to be registered to LTE or 5GNSA mode. Then the test waits for the device to be packet attached and query for the current data-class – at this stage, the current dataclass must be 5G or LTE. After this step, the test connects to the network and validate that the current dataclass is 5G. It may be that 5G_NSA is not shown until the actual 5G carrier is added. Therefore, the test does some download and then checks the current data class; If it is not 5G, this test fails. If the current dataclass is 5G, then the test verifies the signalstate structure. Then disconnect the connection, and set preferred dataclass to LTE to register to LTE mode. Then validate the signalstate structure again.

References : [OID_WWAN_REGISTER_STATE](#) - Windows drivers | Microsoft Learn

[OID_WWAN_PACKET_SERVICE](#) - Windows drivers | Microsoft Learn

[OID_WWAN_CONNECT](#) - Windows drivers | Microsoft Learn

[OID_WWAN_SIGNAL_STATE](#) - Windows drivers | Microsoft Learn

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator that supports 5G NSA.

Test parameters :

AccessString : Specifies the access string to use when making a connection

UserName : Specifies the user name to use when making a connection (optional)

Password : Specifies the password to use when making a connection (optional)

Device.Network.MobileBroadband.NRSA

Mobile broadband

Device.Network.MobileBroadband.NRSA.Discretionary

Mobile broadband devices that implement New Radio Standalone (NRSA) must support exchange interface based on the Mobile Broadband Driver Model.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

Windows Mobile Broadband Driver Model is updated to support 5G SA which include :

- 5G network option 2 (Standalone NR),
- 5G network option 4 (NR and EUTRAN dual connectivity),
- 5G network option 5 (Standalone eLTE),
- 5G network option 7 (Next-gen eLTE and NR dual-connectivity).

Following commands are modified as described in MBIM Extension 3.0 :

- DeviceCapsEx
- ReadyInfo
- ModemConfigInfo
- Registration Parameters
- PacketServiceState (dataClass/dataSubClass should have 5G SA values)

Mobile broadband devices that support this feature must support above commands as specified in MBIM Extension 3.0 document.

NRSA requirement includes the following HLK test cases :

1. Test5GSAExtensions :
This test attempts multiple 5G standalone scenarios :
 - a. The test validates the driver supports DeviceCapsEx complying with 5G SA MBIM Ext 3.0
 - b. The test validates the driver supporting ReadyInfo complying with 5G SA MBIM Ext 3.0
 - c. The test validates the driver supporting Registration/PS complying with 5G SA MBIM Ext 3.0
 - d. The test validates the driver supporting Connect complying with 5G SA MBIM Ext 3.0
 - e. The test validates the driver supporting Connect with WwanActivationOptionPerNonDefaultURSPRules complying with 5G
 - f. The test validates the driver supporting Connect with WwanActivationOptionPerDefaultURSPRules complying with 5G
 - g. The test validates the driver supporting Connect with WwanActivationOptionPerURSPRules complying with 5G

References :

[OID WWAN UE POLICY - Windows drivers | Microsoft Learn](#)
[OID WWAN DEVICE CAPS EX - Windows drivers | Microsoft Learn](#)
[OID WWAN PACKET SERVICE - Windows drivers | Microsoft Learn](#)
[OID WWAN READY INFO - Windows drivers | Microsoft Learn](#)
[OID WWAN CONNECT - Windows drivers | Microsoft Learn](#)

The reader is strongly encouraged to refer the latest MBIM Extended specification to get the updated reference of CIDs corresponding to above OIDs. MBIM spec will always be updated and maintained before the MSDN page is updated.

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator that supports 5G SA. It will be very advantageous if the tester has access to 5G simulator which can simulate 5GSA network. The tester should program the simulator with appropriate set of rules that need to be conveyed to the OS

Test parameters :

AccessString : Specifies the access string to use when making a connection

UserName : Specifies the user name to use when making a connection (optional)

Password : Specifies the password to use when making a connection (optional)

2. TestModemBulkConfig : The test queries modem config info and verifies data members. If modem configuration is pending, verify the later unsolicited notification as well

References :

[OID WWAN MODEM CONFIG INFO - Windows drivers | Microsoft Learn](#)

The reader is strongly encouraged to refer the latest MBIM Extended specification to get the updated reference of CIDs corresponding to above OIDs. MBIM spec will always be updated and maintained before the MSDN page is updated.

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator that supports 5G SA. It will be very advantageous if the tester has access to 5G simulator which can simulate 5GSA network.

Test parameters :

None

3. TestRegistrationParams : This test consists of multiple sub-tests which validate the following:
 - a. The test queries 5G registration parameters and verifies data members
 - b. The test sets 5G registration parameters and verifies data members

- c. The test sets 5G registration parameters with airplane mode (APM) on, turn off APM and verify modem can register with the parameters
- d. The test sets 5G registration parameters and registers device; then sets different registration parameters with forced re-register and verify the device will deregister and re-register

References:

[OID WWAN REGISTER PARAMS - Windows drivers | Microsoft Learn](#)

[OID WWAN MODEM CONFIG INFO - Windows drivers | Microsoft Learn](#)

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator that supports 5G SA. It will be very advantageous if the tester has access to 5G simulator which can simulate 5GSA network.

Test parameters :

None

Device.Network.MobileBroadband.BaseStationInfo.Discretionary

Mobile broadband devices that implement support for providing base station information must support the standardized interface changes based on the Mobile Broadband Driver Model.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

Windows Mobile Broadband Driver Model supports providing location based services with cellular base station information, such as *Base Station ID*, *Time Advance*, and other parameters that can be used to compute the geographical position of the mobile subscriber. The information gathered pertains to the cellular base station currently serving the subscriber, as well as neighboring cellular base stations. Mobile broadband devices that support this feature must be capable of providing base station information for all supported Radio Access Technologies (RATs).

BaseStationInfo requirement has the following HLK test case :

1. TestBaseStationsInfo : This test queries basestations data and verifies data members for modems uptill MBIMEx 2.0
2. TestBaseStationsInfo5GSA : This test queries basestations data and verifies data members for modems from MBIMEx 3.0 and above.

References :

[OID WWAN BASE STATIONS INFO - Windows drivers | Microsoft Learn](#)

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator.

Test parameters :

MaxGSMCount : The maximum number of GSM base stations returned. The default value is 15.

MaxUMTSCount : The maximum number of LTE base stations returned. The default value is 15.

MaxTDSCDMACount : The maximum number of TDSCDMA base stations returned. The default value is 15.

MaxLTECount : The maximum number of LTE base stations returned. The default value is 15.

MaxCDMACount : The maximum number of CDMA base stations returned. The default value is 15.

Device.Network.MobileBroadband.UiccExtensions

Mobile broadband

Device.Network.MobileBroadband.UiccExtensions.LowLevelUiccAccess

Mobile broadband devices that implement low level UICC access must support the low level UICC APDU exchange interface based on the Mobile Broadband Driver Model.

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

Windows Mobile Broadband Driver Model is updated to include the full support of low level APDU exchange with a UICC for getting the Answer-to-Reset (ATR), opening and closing logical channels, generic APDU exchange, and provisioning the Terminal Capability of the device to the UICC.

UiccExtensions.LowLevelUiccAccess requirement has the following HLK test case :

1. TestLowLevelUiccAccess : This test verifies a variety of uicc operations as follows :
 - a. This test queries the supported OIDs from the modem and checks for the necessary support.
 - b. This test attempts to query the ATR and validates the response both for active and inactive slots
 - c. This test attempts to query the terminal capabilities and validates the response both for active and inactive slots
 - d. This test attempts to set the terminal capabilities and validates the response both for active and inactive slots
 - e. This test attempts to open and close a channel and validates the response both for active and inactive slots
 - f. This test checks if ESIM with configured profile is present and execute Apdu command and both for active and inactive slots validates the response
 - g. This test attempts to verify the ready state when no esim profiles are enabled both for active and inactive slots
 - h. This test attempts to Enable a Profile and check the ready state during the transition both for active and inactive slots
 - i. This test attempts to switch a Profile and check the ready state during the transition both for active and inactive slots
 - j. his test attempts to disable a Profile and check the ready state during the transition both for active and inactive slots

References :

[OID_WWAN_UICC_ATR - Windows drivers | Microsoft Learn](#)

[OID_WWAN_UICC_TERMINAL_CAPABILITY - Windows drivers | Microsoft Learn](#)

[OID_WWAN_UICC_RESET - Windows drivers | Microsoft Learn](#)

[MB low level UICC access - Windows drivers | Microsoft Learn](#)

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator. Note that this test do require ESIM to be present in the device and appropriate profiles are present for the ESIM(operational/test profiles).

Test parameters :

None

Device.Network.MobileBroadband.UiccExtensions.AppAccess

Mobile broadband devices that implement UICC app access must support accessing UICC applications and logical file systems based on the Mobile Broadband Driver Model.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Windows Mobile Broadband Driver Model is updated to include the full support of accessing the list of all applications on a UICC, querying the status of specified UICC files, and reading the contents of all UICC file types.

UiccExtensions.AppAccess requirement has the following HLK cases :

1. TestUiccAppAccess : This HLK test has multiple sub-tests which validates the following :
 - a. This test attempts to query the UICC App List and validates the response
 - b. This test attempts to query the file status of EFGID1 and validates the response
 - c. This test attempts to read the value of EFPNN from the UICC and validates the response
 - d. This test attempts to read the value of EFGID1 from the UICC and validates the response
 - e. This test attempts to read the value of EFGID2 from the UICC and validates the response

References :

[OID_WWAN_UICC_APP_LIST - Windows drivers | Microsoft Learn](#)

[OID_WWAN_UICC_FILE_STATUS - Windows drivers | Microsoft Learn](#)

[OID_WWAN_UICC_ACCESS_BINARY - Windows drivers | Microsoft Learn](#)

[OID_WWAN_UICC_ACCESS_RECORD - Windows drivers | Microsoft Learn](#)

[OID_WWAN_PIN_EX2 - Windows drivers | Microsoft Learn](#)

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator.

Test parameters :

None

Device.Network.MobileBroadband.MultiCarrierFunctionality

Mobile broadband devices that support multi-carrier feature must support the multi-carrier functionality.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 ComplyWithBaseReq 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.

MulticarrierFunctionality requirement has the following HLK test case:

1. TestMultiCarrier: The HLK validates that the control capability "MultiCarrier" is supported and if it is supported, it queries and sets the HomeProvider

References : [OID WWAN HOME PROVIDER - Windows drivers | Microsoft Learn](#)

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator.

Test parameters :

MultiCarrierProviderId : Provider Id of the carrier

MultiCarrierProviderName : ProviderName of the Carrier

MultiCarrierCellClass : Cellular class(GSM|CDMA) of the carrier

Device.Network.MobileBroadband.MultiplePDPContext

Multiple PDP context support

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

MultiplePDPContext requirement has the following HLK implemented :

1. TestMPDP : This HLK has multiple sub-tests which validate the following:
 - a. The test verifies the test can successfully connect and disconnect on an additional PDP Context.
 - b. The test connects to the home network via additional PDP context and downloads a file from the internet.
 - c. The test connects to the home network via additional PDP context and default PDP context and downloads a file from the internet.
 - d. The test connects to the home network via additional PDP context and default PDP context and downloads a file from the internet simultaneously.

References :

OID_GEN_DRIVER_VERSION - Windows drivers | Microsoft Learn

OID_WWAN_MPDP - Windows drivers | Microsoft Learn

OID_WWAN_PACKET_SERVICE - Windows drivers | Microsoft Learn

OID_WWAN_CONNECT - Windows drivers | Microsoft Learn

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator – please ensure that the operator and SIM card support creating multiple PDP context connections.

Test parameters :

AccessString : Specifies the access string to use when making a connection

UserName : Specifies the user name to use when making a connection (optional)

Password : Specifies the password to use when making a connection (optional)

WebDoc1024K : Specifies a 1024K Web Doc to download

MpdpAccessString : Specifies the access string to use when making a MPDP connection

MpdpUserName : Specifies the user name to use when making a MPDP connection (optional)

MpdpPassword : Specifies the password to use when making a MPDP connection (optional)

MpdpWebDoc1024K : Specifies a 1024K MPDP Web Doc to download

MpdpIPType : Specifies the IP Type to use when making a MPDP connection

Device.Network.MobileBroadband.SAR

Mobile broadband

Device.Network.MobileBroadband.SAR.Discretionary

Mobile broadband devices that implement support for selective absorption rate (SAR) must support the standardized interface changes based on the Mobile Broadband Driver Model.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

Windows Mobile Broadband Driver model is updated to include support for configuring SAR back off at the device hardware. Devices that implement SAR can optionally support SAR based on the Mobile Broadband Driver Model. If the device supports Mobile Broadband Driver Model SAR, the device firmware needs to support the following at minimum:

- Able to configure SAR back off profile for all antennas
- Provide supported SAR back off mode
- Provide Mobile Broadband and Wi-Fi SAR integration indication
- Provide Mobile Broadband transmission status

SAR requirement has the following HLK implemented :

1. TestSAR : This test implements multiple sub-tests which does the following :
 - a. The test verifies the test can successfully query SAR configurations.
 - b. The test verifies the test can successfully set SAR configurations.
 - c. The test verifies the test can successfully query SAR transmission status.
 - d. The test verifies the test can successfully set SAR transmission status.

References:

OID_WWAN_SAR_CONFIG - Windows drivers | Microsoft Learn

OID_WWAN_SAR_TRANSMISSION_STATUS - Windows drivers | Microsoft Learn

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator. Please note that the test device should support SAR capability.

Test parameters :

None

Device.Network.MobileBroadband.ImplementSMS

The GSM class of mobile broadband devices that implement Short Message Service (SMS) must support SMS based on the Mobile Broadband Driver Model.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Windows Mobile Broadband Driver Model is updated to include the full support of sending and receiving SMS messages. Devices must support SMS based on the Mobile Broadband Driver Model. The device firmware needs to support the following:

- Able to receive and send SMS messages
- Able to delete SMS messages
- Able to configure SMS including SMS Service Center number

HLK test must be performed with a SIM with SMSC configured in it (EF-SMSP 7F10\6F42).

This requirement has the following HLKs implemented:

1. TestSMS : This test has the many subtests which validates the following :
 - a. The test queries sms config and verifies data members.
 - b. The test queries sms status and verifies data members.
 - c. The test sends an SMS message to itself and verifies it receives the message.
 - d. The test deletes all messages.
 - e. The test reads all messages.
 - f. The test reads all new messages.

References:

OID_WWAN_SMS_CONFIGURATION - Windows drivers | Microsoft Learn

OID_WWAN_SMS_DELETE - Windows drivers | Microsoft Learn

OID_WWAN_SMS_READ - Windows drivers | Microsoft Learn

OID_WWAN_SMS_SEND - Windows drivers | Microsoft Learn

OID_WWAN_SMS_STATUS - Windows drivers | Microsoft Learn

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator.

Test parameters :

DevPhoneNumber : Specifies the device's phone number

DevPhoneNumberType : NATIONAL,INTERNATIONAL, or UNKNOWN

2. TestSMSStoreFull : The test verifies proper behavior when the SIM is full.

References :

OID_WWAN_SMS_CONFIGURATION - Windows drivers | Microsoft Learn
OID_WWAN_SMS_DELETE - Windows drivers | Microsoft Learn
OID_WWAN_SMS_READ - Windows drivers | Microsoft Learn
OID_WWAN_SMS_SEND - Windows drivers | Microsoft Learn
OID_WWAN_SMS_STATUS - Windows drivers | Microsoft Learn

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator.

Test parameters :

DevPhoneNumber : Specifies the device's phone number

DevPhoneNumberType : NATIONAL,INTERNATIONAL, or UNKNOWN

Device.Network.MobileBroadband.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 11 Client x64 Windows 11 Client ARM64

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.

USSD requirement has the following HLK test implemented :

1. TestUssd : The test will attempt a USSD set operation and verify the USSD status indication.

References:

[OID_WWAN_USSD - Windows drivers | Microsoft Learn](#)

Test setup : This test requires a Mobile-Broadband supported test device and a data-capable SIM card of any operator.

Test parameters :

UssdStringCodingScheme : Specifies USSD data coding scheme (HexValue) (eg: "0F")

UssdStringChar : Specifies the USSD string to encode (eg: *225#)

UssdStringByte : Specifies the USSD byte string to send (eg: 2A99AC3602)

Device.Network.Switch.Manageability

Device.Network.Switch.Manageability.NetworkSwitchProfile

Device.Network.Switch.Manageability.NetworkSwitchProfile

Applies to**Windows 11 Client x64**

Terms: If-Implemented

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 11 Client 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:

- a. 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.
- b. 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.

Device.Network.WLAN

Windows 11 WLAN drivers can implement either of the following driver models:

- WDF Based NetAdapter Framework model (Wi-FiCx)
- WDI Driver Model

Microsoft will no longer certify any new native Wi-Fi drivers. All new WLAN drivers will need to comply with WDI Architecture or Wi-FiCx architecture to get HLK certified. Previously certified devices with legacy native Wi-Fi drivers will continue to be an option for our partners and will be allowed to recertify with bug fixes/security fixes .

Starting with Windows 11, version 21H2, Microsoft is introducing NetAdapter based driver model for Wi-Fi a.k.a. Wi-FiCx. We expect all drivers to move towards this new model. Wi-FiCx brings in an architecture optimized driver model that will help partners to write simpler drivers while leveraging a lot of capabilities from WDF framework. Newer technology Wi-Fi features such as Wi-Fi 7 will be added only on Wi-FiCx. Only one driver model can be enabled on a Wi-Fi chipset. Microsoft expects new hardware drivers to use Wi-FiCx moving forward. In a future version of the HLK, Microsoft will not certify WDI drivers.

Antenna and Radios

1. For WLAN hardware modules that support Wi-Fi 802.11be (Wi-Fi 7) Microsoft **requires** supporting 2x2 antennas and **recommends** a dual radio design.
2. For WLAN hardware modules that support Wi-Fi 802.11ax (Wi-Fi 6 and 6E) Microsoft **requires** supporting 2x2 antennas and **recommends** a dual radio design.

- a. The 2x2 antenna requirement does not apply to Wi-Fi 6 and 6E devices that are non-mobile such as All-in-Ones (0Dh), Desktop PCs (03h), or Mini-PCs (23h). ¹
3. For WLAN hardware modules that support Wi-Fi 802.11ac (Wi-Fi 5) Microsoft recommends supporting 2x2 antennas and **recommends** a dual radio design
 - a. The 2x2 antenna requirement does not apply to Wi-Fi 5 devices that are non-mobile such as All-in-Ones (0Dh), Desktop PCs (03h), or Mini-PCs (23h). ¹

¹ Please refer to the supported device types on section 1.1.2 of Windows 11 Minimum Hardware Requirements for details. <https://aka.ms/minhw>

Bands

1. Microsoft **requires** support for both the 2.4 GHz and 5 GHz bands and **recommends** support for the 6 GHz band.

Protocol Versions

1. Microsoft **requires** WLAN hardware modules to support at least 802.11ac (Wi-Fi 5)
2. Microsoft **highly recommends** supporting at least 802.11ax (Wi-Fi 6)

The above “highly recommended” items may become requirements for the future release of the Windows Hardware Compatibility Program at Microsoft's discretion.

Device.Network.WLAN.SupportConnectionToAP

Device.Network.WLAN.SupportConnectionToAP.ConnectionToAP (Mandatory)

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

Devices must report supported band information for each respective category. In addition, device must comply with the following requirements.

Driver Servicing

- Driver binary file version must match the version reported in the inf.
- Driver service name must match the driver binary name.

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 Personal networks (WPA2-PSK, WPA3-Personal), 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.

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 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.

12. NLO hint channels
13. Preferred channels
14. 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 where active scanning is allowed - 50ms
- Reconnecting to a channel where 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.

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 802.11v BSS Transition Management Frames will be used to improve roaming performance in the operating system. By utilizing 802.11k Neighbor Reports and 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 for more information.

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.

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:

802.11ax/ac/g/n

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
WPA3-Personal	CCMP

For 802.11ax (if implemented) the following types auth/ciphers must also be supported

Authnetication	Cipher
WPA3-Personal	CCMP

For WPA3 enterprise (if implemented) the following types auth/ciphers must be supported

Authnetication	Cipher
WPA3-Enterprise	GCMP-256

For OWE (if implemented) the following types auth/ciphers must be supported

Authnetication	Cipher
OWE	CCMP

For 802.11ad the following types auth/ciphers must be supported

Authnetication	Cipher
Open	None
WPA2-Enterprise	GCMP
WPA2-Personal	GCMP

[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:

Connectivity:

- Wi-Fi CERTIFIED 6®
- Wi-Fi Direct®
- Wi-Fi Enhanced Open™

Security:

- Protected Management Frames
- WPA2™
- WPA3™

Optimization:

- WMM® (Wi-Fi Multimedia™)

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 PLDR the device either at a bus or at the device level.

System hardware: The system will allow for a PLDR (full device level reset). All systems must support PLDR.

System: The System will indicate support for PLDR 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.

Device.Network.WLAN.SupportConnectionToWiFiAP

Device.Network.WLAN.SupportConnectionToWiFiAP.ConnectionToWiFiAP (Mandatory for all Wi-Fi Radios other than 60GHz)

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description:

Devices must report supported band information for each respective category. In addition, device must comply with the following requirements.

Roam Timing Requirement

For devices that **do not** support Fast Roaming (this means that the STA does not support sending/receiving action frames nor utilizes 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.

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:

Not Applicable for 802.11ad/60GHz/DMG Phy type

- 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 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.

Low Latency Performance

WLAN devices should be able to support low latency applications/scenarios such as Teams/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 ExtSTA 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 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

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 CTS perf measurement tool built into Windows hardware certification kit.

(<https://github.com/microsoft/ctsTraffic>)

- 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.

<u>Channel Width</u>	<u>20 Mhz</u>		<u>40 Mhz</u>		<u>80 Mhz</u>	
<u># of Spatial Streams</u>	<u>Max Phy rate (100%)</u>	<u>Throughput expected by Windows Certification</u>	<u>Max Phy rate (100%)</u>	<u>Throughput expected by Windows Certification</u>	<u>Max Phy rate (100%)</u>	<u>Throughput expected by Windows Certification</u>
<u>1 Stream</u>	86	18	200	42	433	90
<u>2 Stream</u>	173	38	400	88	866	191
<u>3 Stream or higher*</u>	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.

<u>Channel Width</u>	<u>20 Mhz</u>		<u>40 Mhz</u>	
<u># of Spatial Streams</u>	<u>Max Phy rate (100%)</u>	<u>Throughput expected by Windows Certification</u>	<u>Max Phy rate (100%)</u>	<u>Throughput expected by Windows Certification</u>
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.*

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.

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 802.11a (Mandatory)

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 802.11g (Mandatory)

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 (Mandatory)

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 (Mandatory)

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.

Implement 802.11ax (If-Implemented)

Device must report 802.11ax PHY type 10 (WDI_PHY_TYPE_HE) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Implement 802.11be (If-Implemented)

For WiFiCx driver, the driver must report 802.11be PHY type 11 (WDI_PHY_TYPE_EHT) in the call to WifiDeviceSetPhyCapabilities and the driver must also report non-zero values for MaxMLOLinksSupported and MLOAddressesList when setting Station capabilities in the call to WifiDeviceSetStationCapabilities.

Wi-Fi 7 is not supported for WDI drivers.

Device.Network.WLAN.SupportConnectionToWiGigAP

Device.Network.WLAN.SupportConnectionToWiGigAP.ConnectionToWiGigAP
(Mandatory for 60GHz Radio's)

Applies to	Windows 11 Client x64

Description:

Devices must report supported band information for each respective category. In addition, device must comply with the following requirements.

802.11AD Throughput

Refer to Throughput requirements under Device.Network.WLAN.SupportConnectionToWiFiAP for test details

For 802.11ad, the Throughput requirements are as follows:

Throughput Expected for Windows Certification – 1.5 Gbps

Implement 802.11ad (If-Implemented)

Device must report 802.11ad PHY type 9 (WDI_PHY_TYPE_DMG) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Device.Network.WLAN.SupportWakeFromLowPower

Device.Network.WLAN.SupportWakeFromLowPower.WakeFromLowPower

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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/WPA3) and IGTK refresh (WPA2/WPA3) 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).
- When screen is off, device must stay connected and functional after 100 simultaneous power transitions within a stipulated period of time.

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.

Device.Network.WLAN.SupportMACAddressRandomization

Device.Network.WLAN.SupportMACAddressRandomization.MACAddressRandomization (WDI drivers only)

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Network.WLAN.SupportHotspot2Dot0

Device.Network.WLAN.SupportHotspot2Dot0.Hotspot2Dot0 (WDI drivers only)

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

Terms: If-Implemented

Description:

Device must report that it supports ActionFrameSupport in WDI_TLV_INTERFACE_CAPABILITIES and optionally report that it supports ConnectToSSIDsBelongingToHESSID in WDI_STATION_ATTRIBUTES. Please refer to the WDI Spec for implementation details.

Device.Network.WLAN.SupportDot11W

Device.Network.WLAN.SupportDot11W.Dot11W

Applies to

Windows 11 Client x64
Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Network.WLAN.SupportFIPS

Device.Network.WLAN.SupportFIPS.FIPS

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

The requirement does not currently apply to **802.11ad**

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:

1. 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.
2. 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.

WPA3 connections can only be secured with Hardware FIPS. Microsoft **recommends** supporting hardware FIPS

Device.Network.WLAN.SupportWiFiDirect

Device.Network.WLAN.SupportWiFiDirect.WiFiDirect

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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 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

Device.Network.WLAN.SupportWiFiDirectConcurrency

Device.Network.WLAN.SupportWiFiDirectConcurrency.WiFiDirectConcurrency

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Terms: If-Implemented

Support Concurrency

The requirement does not currently apply to **802.11ad**

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

Device.Network.WLAN.SupportWiFiDirectServices

Device.Network.WLAN.SupportWiFiDirectServices.WiFiDirectServices (WDI drivers only)

Applies to	Windows 11 Client x64
	Windows 11 Client ARM64

Terms: If-Implemented

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

Device.Network.WLAN.SupportWirelessDocking

Device.Network.WLAN.SupportWirelessDocking (WDI drivers only)

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description:

Wireless docking solution over 802.11ac or 802.11ad must use Wi-Fi Driver Interface (WDI) driver which provides dock and accessory discovery / connect via native peer to peer and Wi-Fi Direct Services protocol. Please refer to Wi-Fi Driver Interface (WDI) Specification 1.0.20 or later version for feature implementation details

If 802.11ad is being used, the following new PHY type must be reported:

Implement 802.11ad (If-Implemented)

Device must report 802.11ad PHY type 9 (WDI_PHY_TYPE_DMG) in the WDI_PHY_INFO structures returned by OID_WDI_GET_ADAPTER_CAPABILITIES.

Device.Network.WLAN.SupportHostedNetwork

Device.Network.WLAN.SupportHostedNetwork.HostedNetwork (Native Wi-Fi drivers only)

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

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.

Device.Portable.Core

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 11 Client x64

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/?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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64

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 11 Client x64 Windows 11 Client ARM64

Description

A portable device that uses Bluetooth must use the following specifications:

- Bluetooth Core specification Version 2.1 + EDR (with latest errata) or newer [\[https://www.bluetooth.org/Technical/Specifications/adopted.htm\]](https://www.bluetooth.org/Technical/Specifications/adopted.htm)
- MTP over Bluetooth Profile Specification from Microsoft [\[http://msdn.microsoft.com/en-us/windows/hardware/gg463543\]](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 11 Client x64

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_{„t}) 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 11 Client x64

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 11 Client x64

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 11 Client x64

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/?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: · 176Å—144 pixels, 1:1 · 220Å—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 Å—240 pixels, 1:1 · 480 Å—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 Å—480 pixels, 1:1 · 720 Å—480 pixels, 10:11 · 720 Å—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 Å—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

Device.Portable.DigitalCamera

DigitalCamera

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 11 Client x64

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>.

Device.Portable.DigitalVideoCamera

DigitalVideoCamera

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 11 Client x64

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>.

Device.Portable.MediaPlayer

MediaPlayer

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 11 Client x64

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>.

Device.Portable.MobilePhone

MobilePhone

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 11 Client x64

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>.

Device.Stprage

Device.Storage.Controller

General feature that applies to all storage controllers

Device.Storage.Controller.BasicFunction

Storage controller basic functionality

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 PQL. 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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client 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 may 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.

Device.Storage.Controller.Ata

Defines the industry and Microsoft standards that must be met

Device.Storage.Controller.Ata.Interface

PATA Controller Interface

Applies to	Windows 11 Client 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.

Device.Storage.Controller.Boot.BasicFunction

If the controller implements boot support, it must support Int13h functions.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client 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.

Device.Storage.Controller.Fc

Defines the industry and Microsoft standards that must be met

Device.Storage.Controller.Fc.Interface

Fibre Channel HBA interface

Applies to	Windows 11 Client 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.

Device.Storage.Controller.Flush

Defines the industry and Microsoft standards that must be met

Device.Storage.Controller.Flush.BasicFunction

Flush to connected device

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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, NVMe, SCSI and USB), flush cache command shall be sent to connected device without any omission.

Device.Storage.Controller.Iscsi

Defines the industry and Microsoft standards that must be met

Applies to	Windows 11 Client x64

Device.Storage.Controller.Iscsi.Interface

iSCSI interface

Applies to	Windows 11 Client 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 may be implemented and if so, 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 (if implemented), maintain the address of the iSNS server.
- An iSCSI HBA must support changers, disk, tape, and external RAID devices.

Device.Storage.Controller.Optical

General feature that applies to all storage controllers to ensure optical burning requirements are met.

Device.Storage.Controller.Optical.BasicFunction

Storage HBA drivers must support optical drives.

Applies to	Windows 11 Client 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.

Device.Storage.Controller.PassThroughSupport

Pass-through storage controller basic functionality

Device.Storage.Controller.PassThroughSupport.BasicFunction

Pass-through storage controller basic functionality

Applies to	Windows 11 Client 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.

Device.Storage.Controller.Raid

Defines the industry and Microsoft standards that must be met

Device.Storage.Controller.Raid.BasicFunction

RAID controller

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.
- [If Implemented] If the RAID controller supports a disk pass-through mode (aka HBA mode) and the use of SATA drives, it should implement at least the following T10 SAT-4 spec (Rev. 03a) translations:
 - Download Microcode mode 0Eh (Section 8.16.2.5)
 - Download Microcode mode 0Fh (Section 8.16.2.6)

Note: Compliance with this requirement should be tested by connecting a SATA drive, compliant with the Windows 10 firmware update requirements, to the Raid controller in HBA mode and running the HLK Firmware Update Test.

SCSI Requirements can be found in the Device.Storage.SCSI section.

Device.Storage.Controller.Sas

Defines the industry and Microsoft standards that must be met

Device.Storage.Controller.Sas.Interface

SAS controller interface

Applies to	Windows 11 Client 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.

Device.Storage.Controller.Sata

Defines the industry and Microsoft standards that must be met

Device.Storage.Controller.Sata.Interface

SATA controller interface

Applies to	Windows 11 Client x64

Description

SATA Interface

- SATA controllers must comply with the ATA/ATAPI-7 specification.
- SATA controllers must support the hot-plug and surprise removal of target storage devices.
- 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.

Device.Storage.Controller.SD

Defines the industry and Microsoft standards that must be met

Device.Storage.Controller.SD.BasicFunction

SD controller basic functionality

Applies to	Windows 11 Client 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.

Device.Storage.ControllerDrive.NVMe

Storage NVMe feature

Device.Storage.ControllerDrive.NVMe.BasicFunction

NVMe device requirements

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 device must provide physical-location-independent identification information: **(This will be required after September 1, 2017)**

- NVMe devices must use a unique value in the Serial Number field contained within the Identify Controller Data Structure, as described in the NVM Express 1.1b specification
 - The Serial Number field must match the serial number physically printed on a field-replaceable drive.

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 Commit
 - Activation of a firmware image should be done without requiring a power cycle of the device.
 - The activation process is expected to be achieved via a host-initiated reset, as described in section 8.1 of spec version 1.2a.
 - Windows will utilize commit actions 001b or 010b when issuing a firmware commit command.
 - Expected completion values for successful activation without a power cycle are 00h (generic success), 10h, or 11h.
 - If 0Bh is returned as completion status, Windows will inform the user to perform a power cycle of the device. This is highly discouraged, as it prevents updating of firmware at OS runtime and causes significant workload disruption.
- 5.8 Firmware Image Download
 - The device must not fail I/O during the download phase and shall continue serving I/O.
- 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
 - MS (Metadata Size) must be set to 00h, to indicate that no metadata is supported on the current LBA Format.
 - 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.
 - At least NGUID (Namespace Globally Unique Identifier) or EUI64 must be implemented.
 - The controller shall specify a globally unique namespace identifier in this field or the EUI64 field when the namespace is created.
- 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 Format NVM command to perform a cryptographic erase.
 - 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 [If Implemented] 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

Device.Storage.Hd

Device.Storage.Hd.BasicFunction

HD basic functionality

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client 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 11 Client 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 page shall 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

Device.Storage.Hd.1394

Device.Storage.Hd.1394.Compliance

IEEE 1394 hard disk drive specification compliance

Applies to	Windows 11 Client 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

Device.Storage.Hd.Ata

Device.Storage.Hd.Ata.BasicFunction

ATA/ATAPI interface

Applies to	Windows 11 Client 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 11 Client 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.

Device.Storage.Hd.AtaProtocol

Device.Storage.Hd.AtaProtocol.Performance

ATA device performance

Applies to	Windows 11 Client x64

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 11 Client 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.

Device.Storage.Hd.DataVerification

Disk Verification Tests to ensure there is no data loss or corruption

Device.Storage.Hd.DataVerification.BasicFunction

All storage devices must work correctly on Windows.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Storage devices must reliably read and write data without data loss or data corruption.

Device.Storage.Hd.Ehdd

Device.Storage.Hd.Ehdd.Compliance

Encrypted hard drive complies with Microsoft and industry specifications.

Applies to	Windows 11 Client x64

Description

Encrypted hard drive

eDrives must be compliant with these industry specifications

- IEEE 1667 version IEEE 1667-2009 (IEEE 1667-2018 for NVMe devices)
 - 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
- NVMe
 - NVMe 1.1b

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

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

Device.Storage.Hd.EMMC

Defines the industry and Microsoft standards that must be met

Device.Storage.Hd.EMMC.BasicFunction

Emmc basic functionality

Applies to	Windows 11 Client 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.

Device.Storage.Hd.EnhancedStorage

Device.Storage.Hd.EnhancedStorage.1667Compliance

Enhanced Storage devices must comply with the IEEE 1667 defined standards.

Applies to	Windows 11 Client 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/?LinkID=110100>

Device.Storage.Hd.FibreChannel

Device.Storage.Hd.FibreChannel.Compliance

Fibre Channel devices

Applies to	Windows 11 Client 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).

Device.Storage.Hd.Flush

Device.Storage.Hd.Flush.BasicFunction

Flush command completion

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

Device.Storage.Hd.Iscsi

Device.Storage.Hd.Iscsi.BasicFunction

iSCSI devices

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).

Device.Storage.Hd.MultipleAccess.PersistentReservation

Drives that support Persistent Reservations must meet these requirements.

Device.Storage.Hd.MultipleAccess.PersistentReservation.BasicFunction

Drives must provide persistent reservations.

Applies to	Windows 11 Client 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.

Device.Storage.Hd.RaidArray

Device.Storage.Hd.RaidArray.BasicFunction

RAID array systems

Applies to	Windows 11 Client 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 provide redundancy to the failure of at least a single physical disk, and not impair availability of data during rebuilds or drive replacements

RAID arrays implemented as ReFS Backup Targets must be supported by both the HW vendor and Backup Software vendor. Backup Targets include local attached disks via BusTypes SAS, SATA, NVME, RAID, FC, or iSCSI. Backup Targets must comply with industry standard specification requirements with regards to flush functionality:

- For devices utilizing the ATA protocol (including BusType SATA or RAID), 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 devices utilizing the SCSI protocol (including BusType SAS, RAID, FC, or iSCSI), SYNCHRONIZE CACHE (10) command and /or SYNCHRONIZE CACHE (16) command shall be implemented.
- For devices utilizing the NVMe protocol (including BusType NVMe), the device must implement section 6.7 on Flush within the NVMe specification revision 1.0 or sooner.

Device.Storage.Hd.ReadZeroOnTrimUnmap

Device.Storage.Hd.ReadZeroOnTrimUnmap.BasicFunction

The requirement applies to hard disk drives.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

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.

Device.Storage.Hd.RemovableMedia.BasicFunction

Devices with true removable storage media

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

Device.Storage.Hd.Sas

Device.Storage.Hd.Sas.ComplyWithIndustrySpec

Serial attached SCSI devices must comply with industry specifications.

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 LBPM 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.

[Server Only] 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 (subcommand 0Eh) shall not prevent the execution of other read or write I/O in the device queue. No I/O failure is expected during the download operation (subcommand 0Eh)
- If I/O from other initiators is failed during activation (subcommand 0Fh) – it must fail with sense code MICROCODE HAS BEEN CHANGED WITHOUT A RESET.
- 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.

Device.Storage.Hd.Sata

Device.Storage.Hd.Sata.BasicFunction

ATA device performance

Applies to	Windows 11 Client 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

- Recommendation: SATA devices should implement the Device Statistics log (04h) as described in the T13 ATA8-ACS4 specification. In particular, the following pages should be supported:
 - 00h: List of supported log pages
 - 01h: General statistics
 - 03h: Rotating media statistics (for devices that use rotating storage media)
 - 04h: General error statistics
 - 07h: Solid state media statistics
- 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

[Server Only] 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

Device.Storage.Hd.Sata.HybridInformation

This feature is for all devices that support the Hybrid Information feature.

Device.Storage.Hd.Sata.HybridInformation.BasicFunction

SATAIO hybrid drive requirements

Applies to	Windows 11 Client x64

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: $\geq 8 \text{ MB/s @ 4 KB (Queue Depth = 1)}$

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 ($\max(10\%, 0.5 \text{ ms})$). 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 $3/255^{\text{th}}$ 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.

Device.Storage.Hd.Scsi

Device.Storage.Hd.Scsi.Connectors

SCSI connectors

Applies to	Windows 11 Client 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 11 Client 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.

Device.Storage.Hd.Scsi.ReliabilityCounters

Basic reliability counter functionality for disks that implement the SCSI command sets.

Device.Storage.Hd.Scsi.ReliabilityCounters.BasicFunction

Basic reliability counter functionality for disks that implement the SCSI command sets.

Applies to**Windows 11 Client 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)

Device.Storage.Hd.ScsiProtocol

Device.Storage.Hd.ScsiProtocol.ReferenceSpec

Reference to specifications

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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 11 Client x64 Windows 11 Client ARM64

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) as 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.

Device.Storage.Hd.SMART

HDD and SSD storage devices should support Self-Monitoring, Analysis and Reporting Technology (SMART) metrics to reduce the chance of unexpected device failure or data loss.

Device.Storage.Hd.SMART.HealthWarning

Hard drive storage devices should implement SMART critical device health warnings as listed in the ATA8-ACS4 specification.

Applies to

Windows 11 Client x64

Terms: Optional

Enforcement Date: September 1st, 2017

Description

- All HDDs and SSDs:
- Recommendation: support SMART Return Status command as listed in ATA8-ACS4 specification. In particular, the Device Fault Bit (bit 5) in the Status field will be utilized to assess device health.

Device.Storage.Hd.Trim
Device.Storage.Hd.Trim.BasicFunction
ATA Trim and SCSI Unmap functionality

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.

Device.Storage.Hd.Uas
Device.Storage.Hd.Uas.Compliance
USB UAS storage devices

Applies to	Windows 11 Client 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.

Device.Storage.Hd.UasOnEHCI

Device.Storage.Hd.UasOnEHCI.BasicFunction

USB UAS storage devices

Applies to	Windows 11 Client x64

Description

If the device supports UASP on XHCI and then it must support UASP on EHCI.

Device.Storage.Hd.Usb

Device.Storage.Hd.Usb.Compatibility

USB compatibility

Applies to	Windows 11 Client 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

Device.Storage.Hd.Usb3

Device.Storage.Hd.Usb3.Compliance

USB 3.0 storage devices

Applies to	Windows 11 Client 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

Device.Storage.Hd.WindowsToGoCapableUSBDrive

Windows To Go capable USB drive feature

Device.Storage.Hd.WindowsToGoCapableUSBDrive.WindowsToGoCapableUSBDrive

Windows To Go capable USB drive

Applies to	Windows 11 Client 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 WTM 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

Device.Storage.Optical

Device.Storage.Optical.CdRawRecording

Optical drives must support CD RAW recording.

Applies to	Windows 11 Client 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 11 Client 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	
READ ^{10,1,2,4}	500	DVD-RAM / 800
READ ^{12 (not streaming)} ^{1,2,4}	500	DVD-RAM / 800
READ ^{12 (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 11 Client 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 11 Client x64 Windows 11 Client ARM64

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 11 Client 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 11 Client 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 11 Client 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.

Device.Storage.Optical.BluRayReader

Device.Storage.Optical.BluRayReader.Profiles

Required profiles for Blu-Ray readers

Applies to	Windows 11 Client x64

Description

Blu-Ray reader drives must support BD-ROM profile.

Device.Storage.Optical.BluRayWriter

Device.Storage.Optical.BluRayWriter.Profiles

Required profiles for Blu-Ray writers

Applies to	Windows 11 Client x64

Description

Blu-Ray drives that can write must support BD-ROM, BD-R Sequential Recording and BD-RE profiles.

Device.Storage.Optical.Sata

Device.Storage.Optical.Sata.AsynchronousNotification

Asynchronous Notification is required for all SATA connected drives.

Applies to	Windows 11 Client x64

Description

Optical drives that connect via the SATA bus must support Asynchronous Notification.

Device.Streaming.Camera.Base

Device.Streaming.Camera.Base.AVStreamClassInterfaceAndWDM

Camera implementation must be based on AVStream class and WDM, and must meet interface requirements

Applies to	Windows 11 Client x64

Description

Camera implementations 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 register under the class "Camera" in their .inf, not "Image" or "Media."

Camera drivers must support all of the required pins, properties, and settings as defined in the Windows Driver Kit.

Photo Confirmation: Photo Confirmation is now a deprecated control, drivers should no longer support this control.

Camera Profiles (If Implemented):

The driver may advertise profiles which are combinations of camera capabilities that are guaranteed to work simultaneously. Dual use Video and Photo should be excluded from all profiles other than Balanced profile. More information on Camera Profiles can be found in the WDK.

Independent Pins:

All pins advertised by the camera driver 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 of pins for all advertised capabilities for all media types should be supported. If the record and image pins of the camera are not able to stream concurrently, the driver must explicitly advertise this by setting the KSPROPERTY_CAMERACONTROL_IMAGE_PIN_CAPABILITY_EXCLUSIVE_WITH_RECORD and KSPROPERTY_CAMERACONTROL_IMAGE_PIN_CAPABILITY_SEQUENCE_EXCLUSIVE_WITH_RECORD 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](https://msdn.microsoft.com/en-us/library/windows/hardware/jj553707(v=vs.85).aspx)). All other pin combinations must be supported.

USB based cameras must be USB Video Class (UVC) compliant as defined in **Device.Streaming.Camera.UVC.UVCDriver** and **Device.Streaming.Camera.UVC.USBClassDriver**.

Error Conditions

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

Co-installers

Effective from the Windows release after April 1st, 2018, Camera devices must not use co-installers.

Device.Streaming.Camera.Base.MediaFoundation

Camera implementation must support Media Foundation architecture.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Cameras must work with Microsoft Media Foundation.

RGB24 Video Format type is no longer supported. Cameras should remove this mediatype from their descriptors.

Device.Streaming.Camera.Base.MediaTypes

To support increased performance and better visual quality, camera mediatypes must support additional restrictions

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Devices that support MJPEG as a media subtype must either remove the media subtype from its descriptor list, or opt in to pipeline decode of that media subtype on the driver's behalf, using either MS-OS descriptor in firmware or an INF based solution.

Devices may no longer exposed RGB24 as a mediatype

All camera mediatypes must include valid MF_MT_VIDEO_PRIMARY, MF_MT_VIDEO_NOMINAL_RANGE attributes in their mediatype. Additionally, MF_MT_TRANSFER_FUNCTION and MF_MT_YUV_MATRIX attributes are implied but can be overridden. For devices without a DeviceMFT to add these attributes explicitly, an INF solution or UVC MSOS descriptor must be used to add them to the mediatypes.

Device.Streaming.Camera.Base.MultipleClientAppSupport

Camera implementation must support multiple KS filter instances and independent streaming

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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.PnPandPowerPolicies

Camera implementation must comply with Microsoft PNP and power policies.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Cameras must comply with Microsoft PNP and power policies and also support connected stand by configurations.

More info can be found at the following link:

- <http://go.microsoft.com/fwlink/?LinkID=733968>

Device.Streaming.Camera.Base.SurpriseRemoval

Hot-pluggable cameras must support surprise removal.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All hot-pluggable cameras must support their surprise removal from the host bus.

Device.Streaming.Camera.Sensor.KsCategoryRegistration

Non-RGB Cameras must register under KSCategory_Sensor_Camera

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To differentiate the functionality between RGB cameras and other types of sensors, Cameras that are not RGB video streaming cameras must register under category `KSCategory_Sensor_Camera`.

Device.Streaming.Camera.Base.UsageIndicator

Cameras must have a visual indicator to indicate usage

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

A camera must have a visual indicator that indicates when it is streaming. The visual indicator must be on when sensor is capturing data and off when the sensor is not capturing data. If there are multiple cameras on the system but only a single indicator, the indicator must be on when any of the devices are streaming.

A hardware usage indicator (physical LED) is required.

Device.Streaming.Camera.VideoCapture.DirectShow

RGB camera implementation must support DirectShow.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

RGB cameras must work with Microsoft DirectShow in order to support legacy applications. All devices are required to implement DirectShow bridge support.

Device.Streaming.Camera.VideoCapture.KSCategoryRegistration

RGB cameras must register under `KSCategory_Video_Camera`

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

RGB cameras must register under `KSCategory_Video_Camera` for Windows to detect the camera

Device.Streaming.Camera.Base.Sharing

Device.Streaming.Camera.Sensor.FaceAuthMode

To support Face Authentication DDI as supported, Windows Hello additional requirements must also be met

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To advertise that Face Authentication is supported by a device, additional Windows Hello requirements must be met. These include minimum resolutions and frame rates. Please see the Windows Hello Facial Recognition Hardware Specification (V4.5 or above) document for further details.

Device.Streaming.Camera.Base.Sharing.CustomMediaSource

Custom media sources must support the Media Foundation architecture and interfaces

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Cameras and by extension custom media sources that act as cameras must work with Microsoft Media Foundation in order to support Windows applications. In addition to the normal MediaSource interfaces and attributes, the custom media source must be installable as part of a driver package.

Device.Streaming.Camera.Base.Sharing.DeviceMFT

Sharing Camera's DeviceMFT and MFT0 must run out of process as the user mode application

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

To support the sharing experience, the device driver and user mode components associated with the driver (DeviceMFT) must run out of process from the application and must not take a reference to the CaptureEngine object. No more than four DeviceMFTs may be chained.

Support for MFT0 is no longer allowed going forward, and cameras should remove the component or migrate functionality to DeviceMFT.

Device.Streaming.Camera.Base.Sharing.PlatformDeviceMFT

To use Platform Device MFT features, device must support ROI control

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Devices that opt in to Platform Device MFT must have ROI driver control supported. ROI must be supported through Extended Region of Interest properties: KSPROPERTY_CAMERACONTROL_EXTENDED_ROI_CONFIGCAPS and KSPROPERTY_CAMERACONTROL_EXTENDED_ROI_ISPCONTROL.

Device.Streaming.Camera.Base.Sharing.MemoryAllocation

Memory allocation by drivers must be aligned to 4k (4096 bytes) page boundaries.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Memory allocation by drivers must be aligned to 4k page boundaries. It is required that the driver's user mode component use the the Camera Pipeline allocator

Device.Streaming.Camera.Base.Sharing.NonConsumableData

Streams not meant for user consumption must be hidden

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Any stream exposed by the driver should be usable in some way by an application. The data type for the stream being well understood is not required, but the stream should be functional in all other aspects of streaming. If the data from a stream cannot be consumed by another component or application, this must be marked on the pin so that it will be hidden from applications.

Device.Streaming.Camera.Base.HardwarePrivacy

Device.Streaming.Camera.Base.HardwarePrivacy

Cameras that implement physical hardware shutters will use supported mechanisms to inform the OS.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Cameras that provide mechanisms for customers to occlude or disable use of the camera must:

- Use an allowable mechanism: manual shutter, electromechanical shutter controlled by a hotkey/switch, or hotkey/switch that acts as a kill switch by blocking transfer of captured image data out of the ISP (e.g., replaces it with synthesized black frames or similar). Hotkey- or switch-based solutions must use physical controls that are implemented entirely through firmware.
- Report state of the shutter/kill switch whenever it is changed while the device is actively streaming, through KSPROPERTY_CAMERACONTROL_PRIVACY.

- **Not** change state of the mechanism due to any software actions or commands, including Companion Apps, custom or standard APIs, or camera streaming state changes. Mechanism state changes must strictly be controlled by physical interaction with the shutter or the system’s hotkey/switch.
- **Not** interrupt the ability for applications to enumerate or use the camera while the shutter/kill switch is active. Frames must continue to be delivered to apps streaming from the camera, and shutter/kill switch state changes while streaming must not interrupt the flow of frames.
- Light the Usage Indicator LED whenever applications are streaming from the camera and the shutter is closed. However, a shutter’s mechanical design may occlude visibility of the Usage Indicator LED when active.
- Light the Usage Indicator LED whenever applications are streaming from the camera and the camera is not disabled by a kill switch.
- Prevent capture of images from **all** cameras on the same panel (e.g., RGB and IR) when a shutter on that panel is closed. It is acceptable if a shutter physically occludes only one of multiple cameras on a panel (e.g., the RGB camera) if the ISP applies firmware-based image occlusion to other cameras on the panel (e.g., the IR camera) when the shutter is active.
- Prevent captured images from all cameras within the chassis (e.g., RGB and IR; front-facing and world-facing) from streaming out of the ISP when a kill switch is activated.

In the context of kill switches, “active” refers to a switch or hotkey latched in the “off” position. In the context of shutters, “active” refers to the position that physically occludes the camera sensor.

Please refer to [Camera privacy shutters and kill switches](#) for further design details and guidance.

Device.Streaming.Camera.Sharing.VirtualCamera

Virtual cameras should support all base camera streaming scenarios

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Virtual Cameras packaged as part of the camera driver must stream as a camera source. It cannot be used as part of FaceAuth or Secure streaming. When tied to physical camera hardware, they must declare which physical camera hardware(s) they are associated with and that associated physical hardware must not be hidden.

Device.Streaming.Camera.Sharing.CameraAIEffects

Cameras that opt into MSFT Camera AI Effects must support streaming with effects enabled/disabled

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Camera that opt into AI Effects should be able to stream with both effects enabled and disabled.

Device.Streaming.Camera.UVC

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 11 Client x64 Windows 11 Client ARM64

Description

All USB streaming video Cameras must be compatible with Microsoft USB Video class driver (USBVideo.sys). All USB streaming video cameras are expected to work without custom components, such as DeviceMFT, as simple cameras

Effective from the Windows release after April 1st, 2018, all USB streaming video Cameras must use the Microsoft USB Video class driver (USBVideo.sys). Custom drivers are not allowed. Any custom functionality must be implemented by creating a Device MFT (preferred) or by implementing extension units.

Device.Streaming.Camera.UVC.UVCCacheDeprecation

UVC Drivers may no longer support UVC Cache Controls

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Devices may no longer implement UVC Cache controls. This is now a deprecated feature.

Device.Streaming.Camera.UVC.UVCDriver

USB streaming video cameras must comply with USB Video Class specifications

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

All USB streaming video cameras 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.

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>. In addition, the H.264 bitstream must be compliant with Media Foundation.

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

Device.Streaming

Device.Streaming.D3D9.H264

Device.Streaming.D3D9.H264.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of H264 content. Only required if implemented.

Device.Streaming.D3D9.MPEG2

Device.Streaming.D3D9.MPEG2.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of of MPEG2 content. Only required if implemented.

Additional Information

[MSDN Media Foundation Programming Guide](#)

[DXVA Video Processing Documentaiton](#)

[DXVA High Definition Documentation](#)

[DXVA 2.0 Video Acceleration Documentation](#)

Device.Streaming.D3D9.VC1

Device.Streaming.D3D9.VC1.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of of VC1 content. Only required if implemented.

Additional Information

[VC1 \(also known as MPEG-2 or WMV\) DXVA Specifications](#)

[MSDN Media Foundation Programming Guide](#)
[DXVA Video Processing Documentaiton](#)
[DXVA High Definition Documentation](#)
[DXVA 2.0 Video Acceleration Documentation](#)

Device.Streaming.D3D11.HDRStreaming

Device.Streaming.D3D11.HDRStreaming.VP9

This feature enables the detection of devices which have hardware support for EDR with VP9 content.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

This feature will allow the HLK to appropriately gate Streaming HDR (also referred to as 'EDR') tests against hardware that can support it

Device.Streaming.D3D11.HDRStreaming.HEVC

This feature enables the detection of devices which have hardware support for EDR with HEVC content.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description:

This feature will allow the HLK to appropriately gate Streaming HDR (also referred to as 'EDR') tests against hardware that can support it.

Device.Streaming.D3D11.AV1.Profile0.4k.Playback

4k AV1 Profile 0 Hardware Playback should be supported if there is hardware support for 4k AV1 Profile 0

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

DXVA accelerated decode of 4k AV1 content. This requirement explicitly checks for 4k support. Only required if implemented.

Device.Streaming.D3D11.H264

Device.Streaming.D3D11.H264.4k.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of 4k H264 content. This requirement explicitly checks for 4k support. Only required if implemented.

Device.Streaming.D3D11.HDRStreaming

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

Streaming HDR may also be referred to EDR (Enhanced Dynamic Rendering).

For a device to be considered capable of HDR Streaming it must have:

- A max luminance ≥ 300 nits
- Capable of resolutions greater than 1920x1080

Device.Streaming.D3D11.HDRStreaming.HEVC.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

HDR streaming DXVA playback with 10-bit HEVC content. May also be referred to EDR (Enhanced Dynamic Rendering).

Device.Streaming.D3D11.HDRStreaming.VP9.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

HDR streaming DXVA playback with 10-bit VP9 content. HDR streaming.

Additional Information

[MSDN Media Foundation Programming Guide](#)

Device.Streaming.D3D11.HEVC

Device.Streaming.D3D11.HEVC.Main.4k.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of 8-bit 4k HEVC content. This requirement explicitly checks for 4k support. Only required if implemented.

Device.Streaming.D3D11.HEVC.Main.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of 8-bit HEVC content.

Device.Streaming.D3D11.HEVC.Main10.4k.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of 10-bit 4k HEVC content. This requirement explicitly checks for 4k support. Only required if implemented.

Device.Streaming.D3D11.HEVC.Main10.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of 10-bit HEVC content.

Device.Streaming.D3D11.VP8

Device.Streaming.D3D11.VP8.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of VP8 content. Only required if implemented.

Additional Information

[VP8/VP9 DXVA Specifications](#)
[MSDN Media Foundation Programming Guide](#)
[DXVA Video Processing Documentaiton](#)
[DXVA High Definition Documentation](#)
[DXVA 2.0 Video Acceleration Documentation](#)

Device.Streaming.D3D11.VP9

Device.Streaming.D3D11.VP9.Profile0.4k.Playback

Applies to	Windows 11 Client x64

Windows 11 Client ARM64

Description

DXVA accelerated decode of 8-bit 4k VP9 content. This requirement explicitly checks for 4k support. Only required if implemented.

Device.Streaming.D3D11.VP9.Profile0.Playback**Applies to****Windows 11 Client x64****Windows 11 Client ARM64**

Description

DXVA accelerated decode of 8-bit VP9 content. Only required if implemented.

Device.Streaming.D3D11.VP9.Profile2.Playback**Applies to****Windows 11 Client x64****Windows 11 Client ARM64**

Description

DXVA accelerated decode of 10-bit VP9 content. Only required if implemented.

Device.Streaming.D3D12.H264.Playback**Applies to****Windows 11 Client x64****Windows 11 Client ARM64**

Description

DXVA accelerated decode of H264 content. Only required if implemented.

Additional Information[H264/AVC DXVA Specifications](#)[MSDN Media Foundation Programming Guide](#)[DXVA Video Processing Documentaiton](#)[DXVA High Definition Documentation](#)[DXVA 2.0 Video Acceleration Documentation](#)**Device.Streaming.D3D12.HEVC****Device.Streaming.D3D12.HEVC.Main.4k.Playback****Applies to****Windows 11 Client x64****Windows 11 Client ARM64**

Description

DXVA accelerated decode of 8-bit 4k HEVC content. This requirement explicitly checks for 4k support. Only required if implemented.

Device.Streaming.D3D12.HEVC.Main.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of 8-bit HEVC content. Only required if implemented.

Device.Streaming.D3D12.HEVC.Main10.4k.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of 10-bit 4k HEVC content. This requirement explicitly checks for 4k support. Only required if implemented.

Additional Information

[H265 \(HEVC\) DXVA Specifications](#)

[MSDN Media Foundation Programming Guide](#)

[DXVA Video Processing Documentaiton](#)

[DXVA High Definition Documentation](#)

[DXVA 2.0 Video Acceleration Documentation](#)

Device.Streaming.D3D12.AV1.Profile0

This section contains requirements for D3D12 AV1 Profile 0 Playback

Device.Streaming.D3D12.AV1.Profile0.Playback

AV1 Profile 0 Hardware Playback should be supported if there is hardware support for AV1 Profile 0

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

D3D12 DXVA accelerated decode of AV1 content. Only required if implemented.

Device.Streaming.D3D12.AV1.Profile0.4k.Playback

4k AV1 Profile 0 Hardware Playback should be supported if there is hardware support for 4k AV1 Profile 0

Applies to	Windows 11 Client x64

Terms: If-Implemented

Description

D3D12 DXVA accelerated decode of 4k AV1 content. This requirement explicitly checks for 4k support. Only required if implemented.

Device.Streaming.HMFT

Device.Streaming.HMFT.Decoding

Hardware Media Foundation Transform (HMFT) must support video decoding.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 in-box, 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 11 Client x64 Windows 11 Client ARM64

Description

HMFT video encoding is for each format is only required if-implemented.

A. VC-1 Encode

If your hardware supports VC-1 Encode, you must:

A.1 On input:

A.1.1 Support NV12

A.1.2 If your hardware supports it:

A.1.2.1 Support IYUV and YUY2

A.1.3 Support input buffers of (and query in this order):

A.1.3.1 IMF2DBuffer2

A.1.3.2 [IMF2DBuffer](#)

A.1.3.3 [IMFMediaBuffer](#)

A.2 On output:

A.2.1 Support Simple profile

A.2.2 Support Main profile

A.2.3 Support Advanced profile

A.3 Your VC-1 encoder must expose the following interfaces:

A.3.1 [IMFTransform](#)

A.3.2 [ICodecAPI](#)

A.3.2.1 ICodecAPI requires the following functions to be implemented:

A.3.2.1.1 [IsSupported](#)

A.3.2.1.2 [GetValue](#)

A.3.2.1.3 [SetValue](#)

A.3.2.2 ICodecAPI requires the following properties to be supported:

A.3.2.2.1 Peak-constrained VBR mode

A.3.2.2.2 Quality-based VBR mode

A.3.2.2.3 CBR encoding

A.3.2.2.4 GOP distance

A.3.2.2.5 Frame-QP control

A.3.2.2.6 Adaptive bitrate control

A.3.2.2.7 Force-Key-frame control

A.3.2.2.8 QualityVsSpeed control

A.3.2.2.9 Low-latency encoding

A.3.2.3 It is recommended that you additionally implement the following functions:

A.3.2.3.1 [GetDefaultValue](#)

A.3.2.3.2 [GetParameterRange](#)

A.3.2.3.3 [GetParameterValues](#)

A.3.2.3.4 [IsModifiable](#)

A.3.2.3.5 [SetAllDefaults](#)

A.3.3 [IMFAttributes](#)

A.3.3.1 Via [IMFTransform::GetAttributes](#)

A.3.3.2 The two required attributes are:

A.3.3.2.1 [MFT_ENUM_HARDWARE_URL_Attribute](#)

A.3.3.2.2 [MFT_ENUM_HARDWARE_VENDOR_ID_Attribute](#)

A.3.4 It is recommended that you implement an [asynchronous MFT](#).

A.3.4.1 Asynchronous MFTs require the following additional interfaces:

A.3.4.1.1 [IMFMediaEventGenerator](#)

A.3.4.1.2 [IMFShutdown](#)

A.3.4.2 Asynchronous MFTs are encouraged to avoid creating threads and are recommended to use the MF Thread Pool.

A.3.4.2.1 Registration with MMCS via IMFRealTimeClientEx::SetWorkQueue is critical to meet performance goals.

A.3.5 IMFRealTimeClientEx is recommended

A.4 Encoding Settings

A.4.1 Through input media type negotiation, the VC-1 Encoder must support:

A.4.1.1 [MF_MT_INTERLACE_MODE](#)

A.4.1.1.1 The encoder must preserve interlace from input to output or reject interlace.

A.4.1.2 [MF_MT_MINIMUM_DISPLAY_APERTURE](#)

A.4.2 Through output media type negotiation, the VC-1 Encoder must support:

A.4.2.1 [MF_MT_SUBTYPE](#)

A.4.2.2 [MF_MT_FRAME_SIZE](#)

A.4.2.3 [MF_MT_FRAME_RATE](#)

A.4.2.4 [MF_MT_AVG_BITRATE](#)

A.4.2.5 [MF_MT_PIXEL_ASPECT_RATIO](#)

A.4.3 It is recommended that your VC-1 Encoder supports:

A.4.3.1. B frame encoding

A.5 Multiple Instances

A.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.

A.5.1.1 These instances may be in the same process or in different processes.

A.6 Merit Validation

A.6.1 It is required that your VC-1 encoder supports the trusted merit verification process.

A.6.2 It is required that your VC-1 encoder be a separate binary, individually certified and signed.

A.7 Additional Requirements

A.7.1 Your VC-1 encoder must work with the [Windows ASF file sink](#).

A.7.2 Your VC-1 encoder must implement proper order of encoding configuration.

A.7.3 Your VC-1 encoder must work correctly in session 0.

A.8 Installation

A.8.1 It is required that your VC-1 encoder is registered and unregistered along with the device driver used in the encoder.

Device.Streaming.HMFT.H264.Encoding

Hardware Media Foundation Transform (HMFT) must support video encoding.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

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 bitstreams 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.

Device.Streaming.HMFT.H265.Encoding

Hardware Media Foundation Transform (HMFT) must support video encoding.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

A. H.265 Encode

If your hardware supports H.265 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. For encoders that support H.265 Main10 profile, P010 shall be supported.

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 Main profile(and Main10 if supported).

The encoder must be capable of generating the appropriate header syntax elements to signal that it is encoded to the Main profile.

A.3 Interface Support Requirements

The H.265 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.3.2.2.10 CODECAPI_AVEncVideoForceKeyFrame

A.3.2.2.11 CODECAPI_AVEncVideoEncodeQP

A.3.2.3 Optional Properties

Support for the following ICodecAPI property GUIDs is optional.

A.3.2.3.1 CODECAPI_AVEncCommonQualityVsSpeed

A.3.2.3.2 CODECAPI_AVEncVideoTemporalLayerCount

A.3.2.3.3 CODECAPI_AVEncVideoSelectLayer

A.3.2.3.4 CODECAPI_AVEncMPVDefaultBPictureCount

A.3.2.3.5 CODECAPI_AVEncVideoEncodeFrameTypeQP

A.3.2.3.6 CODECAPI_AVEncSliceControlMode

A.3.2.3.7 CODECAPI_AVEncSliceControlSize

A.3.2.3.8 CODECAPI_AVEncVideoMaxNumRefFrame

A.3.2.3.9 CODECAPI_AVEncVideoMeanAbsoluteDifference

A.3.2.3.10 CODECAPI_AVEncVideoROIEnabled

A.3.2.3.11 CODECAPI_AVEncVideoMinQP

A.3.2.3.12 CODECAPI_AVEncVideoMaxQP

A.3.2.3.13 CODECAPI_AVEncVideoLTRBufferControl

A.3.2.3.14 CODECAPI_AVEncVideoMarkLTRFrame

A.3.2.3.15 CODECAPI_AVEncVideoUseLTRFrame

A.3.2.3.16 CODECAPI_AVEncLowPowerEncoder

A.3.2.3.17 CODECAPI_AVScenarioInfo

A.3.2.3.18 CODECAPI_AVEncMPVGOPSizeMin

A.3.2.3.19 CODECAPI_AVEncMPVGOPSizeMax

A.3.2.3.20 CODECAPI_AVEncVideoMaxTem

A.3.2.3.21 CODECAPI_AVEncMaxFrameRate

A.3.2.3.22 CODECAPI_VideoEncoderDisplayContentType

A.3.2.3.23 CODECAPI_AVEncEnableVideoProcessing

A.3.2.3.24 CODECAPI_AVEncVideoGradualIntraRefresh

A.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.

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](#)

The following attributes are required to be supported only if the encoder supports the Main10 profile

A.4.17 [MF_MT_MAX_MASTERING_LUMINANCE](#)

A.4.18 [MF_MT_MIN_MASTERING_LUMINANCE](#)

A.4.19 [MF_MT_CUSTOM_VIDEO_PRIMARIES](#)

A.4.20 [MF_MT_MAX_LUMINANCE_LEVEL](#)

A.4.21 [MF_MT_MAX_FRAME_AVERAGE_LUMINANCE_LEVEL](#)

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

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.

A.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;
```

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.265 encoder must support the following functionality:

A.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.

A.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.

A.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).

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 bitstreams of 1, 2, or 3 temporally-independent layers.

A.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

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.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.

A.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.

A.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.

A.12 Installation

It is required that your H.265 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.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.

Device.Streaming.HMFT.VP8.Encoding

Hardware Media Foundation Transform (HMFT) must support video encoding.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

A. VP8 Encode

If your hardware supports VP8 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 Profile0.

The encoder must be capable of generating the appropriate header syntax elements to signal that it is encoded to the Profile0.

A.3 Interface Support Requirements

The VP8 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.3.2.2.10 CODECAPI_AVEncVideoForceKeyFrame

A.3.2.2.11 CODECAPI_AVEncVideoEncodeQP

A.3.2.3 Optional Properties

Support for the following ICodecAPI property GUIDs is optional.

A.3.2.3.1 CODECAPI_AVEncCommonQualityVsSpeed

A.3.2.3.2 CODECAPI_AVScenarioInfo

A.3.2.3.3 CODECAPI_AVEncMPVGOPSizeMin

A.3.2.3.4 CODECAPI_AVEncMPVGOPSizeMax

A.3.2.3.5 CODECAPI_VideoEncoderDisplayContentType

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.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.5.2 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

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.

A.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;
```

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 VP8 encoder must support the following functionality:

A.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.

A.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.

A.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).

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.

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 VP8 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.

A.10 Merit Validation

It is required that your VP8 encoder supports the trusted merit verification process.

It is required that your VP8 encoder be a separate binary, individually certified and signed.

A.11 Additional Requirements

Your VP8 encoder must implement proper order of encoding configuration.

Your VP8 encoder must send one batched GPU request per frame.

Your VP8 encoder must work correctly in session 0.

A.12 Installation

It is required that your VP8 encoder is registered and unregistered along with the device driver used in the encoder.

A.113 Dynamic Format Change

It is required that your VP8 encoder supports dynamic format changes of the following within:

Profile Change

Resolution and Frame Rate Change

When multiple changes are requested before an output frame is produced, the last change should be honored.

Device.Streaming.HMFT.VP9.Encoding

Hardware Media Foundation Transform (HMFT) must support video encoding.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

A. VP9 Encode

If your hardware supports VP9 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 Profile0.

The encoder must be capable of generating the appropriate header syntax elements to signal that it is encoded to the Profile0.

A.3 Interface Support Requirements

The VP9 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.3.2.2.10 CODECAPI_AVEncVideoForceKeyFrame

A.3.2.2.11 CODECAPI_AVEncVideoEncodeQP

A.3.2.3 Optional Properties

Support for the following ICodecAPI property GUIDs is optional.

A.3.2.3.1 CODECAPI_AVEncCommonQualityVsSpeed

A.3.2.3.2 CODECAPI_AVScenarioInfo

A.3.2.3.3 CODECAPI_AVEncMPVGOPSizeMin

A.3.2.3.4 CODECAPI_AVEncMPVGOPSizeMax

A.3.2.3.5 CODECAPI_VideoEncoderDisplayContentType

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.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.5.2 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

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.

A.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;
```

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 VP9 encoder must support the following functionality:

A.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.

A.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.

A.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).

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.

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 VP9 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.

A.10 Merit Validation

It is required that your VP9 encoder supports the trusted merit verification process.

It is required that your VP9 encoder be a separate binary, individually certified and signed.

A.11 Additional Requirements

Your VP9 encoder must implement proper order of encoding configuration.

Your VP9 encoder must send one batched GPU request per frame.

Your VP9 encoder must work correctly in session 0.

A.12 Installation

It is required that your VP9 encoder is registered and unregistered along with the device driver used in the encoder.

A.113 Dynamic Format Change

It is required that your VP9 encoder supports dynamic format changes of the following within:

Profile Change

Resolution and Frame Rate Change

When multiple changes are requested before an output frame is produced, the last change should be honored.

Device.Streaming.HMFT.AV1

Device.Streaming.HMFT.AV1.Encoding

HMFT hardware encoding of AV1 content

Applies to	Windows 11 Client x64

If Implemented

Description

Validates compliance to AV1 HW-MFT specification to enable IHVs to ship hardware accelerated AV1 video encoding support. IHVs shall implement an MFT capable of AV1 profile 0 encode support, supporting NV12 and P010 input formats. This MFT must support the IMFTTransform and ICodecAPI interfaces. And via these interfaces provide support for: peak-constrained VBR rate control, quality based VBR rate control, CBR rate control, GOP distance control, frame-based QP control and forced keyframe control. *Only required if implemented.*

Device.Streaming.HMFT.Displayable.MJPEG

Device.Streaming.HMFT.Displayable.MJPEG.Decoding

Output displayable resources from a HMFT hardware decoder of MJPEG content.

Applies to

Windows 11 Client x64

Description

Output displayable resources from a HMFT hardware decoder of MJPEG content. This requirements checks for a MJPEG HMFT GUID and the satisfaction of Device.Graphics.WDDM30.DisplayRender.FlexibleDisplayableSurfacePresentation. This requirement allows the HMFT decoder and the MediaFoundation video rendering pipeline to participate in direct presentation with the composition swapchain API's. 'Direct presentation' refers to the ability to present the same output buffer provided by the decoder without requiring an intermediate copy from the decoder output into a swapchain resource.

If implemented and the MF_ATTRIBUTE 'MF_SA_D3D11_ALLOCATE_DISPLAYABLE_RESOURCES

' is set on the HMFT hardware decoder then the HMFT hardware decoder must provide displayable output samples. At the time of writing the documentation there is no publicly available version of the documentation describing the allocation of displayable output samples in a decoder MFT. When this documentation is published you will be able to search for this new attribute on MSDN.

It is only required if a MJPEG HMFT is implemented on a device driver which supports Device.Graphics.WDDM30.DisplayRender.FlexibleDisplayableSurfacePresentation

Device.Streaming.HMFT.H263

Device.Streaming.HMFT.H263.Decoding

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

HMFT support for hardware decode of H263 content. This requirements checks for a H263 HMFT GUID.

Device.Streaming.HMFT.HEVC

Device.Streaming.HMFT.HEVC.Encoding

Applies to

Windows 11 Client x64

Windows 11 Client ARM64

Description

HMFT support for hardware encode of HEVC content. This requirements checks for a HEVC Encode HMFT GUID. Only required if implemented.

Device.Streaming.HMFT.HEVC.MAIN10.Encoding

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description

HMFT support for hardware encode of HEVC 10-bit content. This requirements checks for a HEVC 10-bit Encode HMFT GUID. Only required if implemented.

Device.Streaming.HMFT.MJPEG

Device.Streaming.HMFT.MJPEG.Decoding

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description

HMFT support for hardware decode of MJPEG content. This requirements checks for a MJPEG HMFT GUID. Only required if implemented.

Device.Streaming.HMFT.MPEG4

Device.Streaming.HMFT.MPEG4.Decoding

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description

HMFT support for hardware decode of MPEG4 content. This requirements checks for a MPEG4 HMFT GUID. Only required if implemented.

Device.Streaming.HWDRM.H264

Device.Streaming.HWDRM.H264.CBCS.Playback

Applies to**Windows 11 Client x64****Windows 11 Client ARM64**

Description

HWDRM support for DXVA accelerated decode of H264 content. Using the AES-CBC Encryption scheme.

Specifically checks for the 'D3D11_DECODER_BITSTREAM_ENCRYPTION_TYPE_CBCS' Bitstream Encryption GUID.

Only required if implemented.

Device.Streaming.HWDRM.H264.CENC.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

HWDRM support for DXVA accelerated decode of H264 content. Using Common Encryption (CENC).

Specifically checks for the 'D3D11_DECODER_BITSTREAM_ENCRYPTION_TYPE_CENC' Bitstream Encryption GUID.

Only required if implemented.

Device.Streaming.HWDRM.H264.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of H264 content.

Specifically checks for the 'D3D11_DECODER_ENCRYPTION_HW_CENC' Bitstream Encryption GUID.

Only required if implemented.

Device.Streaming.HWDRM.HEVC

Device.Streaming.HWDRM.HEVC.Main.CENC.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

HWDRM support for DXVA accelerated decode of HEVC 8-bit content. Using Common Encryption (CENC).

Specifically checks for the 'D3D11_DECODER_BITSTREAM_ENCRYPTION_TYPE_CENC' Bitstream Encryption GUID.

Only required if implemented.

Device.Streaming.HWDRM.HEVC.Main.Playback

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Description

DXVA accelerated decode of HEVC 8-bit content. Explicitly checks that a device is not ParaVirtualized.

Specifically checks for the 'D3D11_DECODER_ENCRYPTION_HW_CENC' Bitstream Encryption GUID.

Only required if implemented.

Device.Streaming.HWDRM.HEVC.Main10.Playback

Applies to	Windows 11 Client x64
	Windows 11 Client ARM64

Description

DXVA accelerated decode of HEVC 10-bit content.

Specifically checks for the 'D3D11_DECODER_ENCRYPTION_HW_CENC' Bitstream Encryption GUID.

Only required if implemented.

Device.Streaming.HWDRM.VP9

Device.Streaming.HWDRM.VP9.Profile2.10bit.CENC.Playback

Applies to	Windows 11 Client x64
	Windows 11 Client ARM64

Description

HWDRM support for DXVA accelerated decode of VP9 10-bit content. Using Common Encryption (CENC).

Specifically checks for the 'D3D11_DECODER_BITSTREAM_ENCRYPTION_TYPE_CENC' Bitstream Encryption GUID.

Device.Streaming.HWDRM.AV1.Profile0

This section contains requirements for HWDRM AV1 Profile 0 Playback

Device.Streaming.HWDRM.AV1.Profile0.Playback

AV1 Profile 0 Hardware Playback should be supported if there is hardware support for AV1 Profile 0

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

HWDRM playback of AV1 content. Only required if implemented.

Device.Streaming.HMFT.Displayable.MJPEG

Device.Streaming.HWDRM.AV1.Profile0.CBCS.Playback

AV1 Profile 0 Hardware Playback should be supported if there is hardware support for AV1 Profile 0

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

HWDRM cbcS playback of AV1 content. Only required if implemented.

Device.Streaming.HWDRM.AV1.Profile0.CENC.Playback

AV1 Profile 0 Hardware Playback should be supported if there is hardware support for AV1 Profile 0

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Terms: If-Implemented

Description

HWDRM cenc playback of AV1 content. Only required if implemented.

Device.TrustedPlatformModule

Device.TrustedPlatformModule.TPM20

Informative: TPM devices are tested separately from system submissions. System submissions will require a TPM which is considered certified at the time of system certification. TPM devices which obtain certification under these requirements will generally be considered acceptable for all versions of Windows for a period of up to 3 years from the time of certification – based upon the HLK Version of the submission. This period is extended for 1 year, for a total of 4 years, for Server devices. The policies are enforced at Microsoft's sole discretion.

Informative: The TPM is a hardware security device, and Windows relies on the security of the TPM to provide security guarantees for Windows features. Accordingly, Microsoft may at its sole discretion, revoke the Windows certification of a specific firmware version or hardware model as appropriate.

Device.TrustedPlatformModule.TPM20.Features

All TPM 2.0 Devices must support the features specified in this section.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Mandatory: The TPM must comply with:

1. TCG Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 1.59, including Errata Version 1.1 or a newer published Revision and all security patches published at least 120 days prior to testing.

Mandatory: The TPM 2.0 must implement all "Mandatory" features in the table below:

Integrity Feature	SOC Hardware Functionality	Status
Trusted Execution Environment	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. Availability of access to persistent storage prior to a call to ExitBootServices.	Mandatory
Secure Storage	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.	Mandatory
Platform Attestation	Implementation of PCRs 0 through 23 for SHA-256.	Mandatory
	Implementation of PCRs 0 through 23 for SHA-384 Implementation of PCRs 0 through 23 for SHA-384 becomes mandatory with the First Windows Release in 2025.	Recommended
Side Channel Protection	Robustness against side channel attacks including Differential Power Analysis (DPA) and Electromagnetic Emanations (EM)	Recommended
Miscellaneous	Execution of the TPM 2.0 command TPM2_NV_Increment must not require an open object slot.	Mandatory

Under normal use cases the clock may not become “Unsafe”. Normal use cases exclude a “disorderly shutdown” as defined in TCG TPM Library Specification Part 1.

The clock must run at all times when power is applied to the TPM.

The TPM may only increment the FailedTries counter on TPM_Startup if no TPM_Shutdown has occurred since the most recent use of a dictionary attack protected asset.

The TPM Vendor must provide Microsoft with documentation detailing how to decode TPM Get Capability calls into the TPM Manufacturer Identity, Model Number and Firmware Version. All of these items must be available on every device, and consistent with the actual firmware and model under test.

The TPM must support TPM_PT_NV_BUFFER_MAX of at least 0x300 (HEX) bytes.

The TPM Vendor must on request provide sample devices to Microsoft to validate compliance with the requirements before being considered certified.

Mandatory: The TPM must comply with “TCG PC Client Platform TPM Profile (PTP) Specification”, Family “2.0”, Level 00 Revision 1.04 or later, except as stated below:

- Any statements in the “System.Fundamentals.TPM20” or “Device.TrustedPlatformModule.TPM20” are authoritative if there is a conflict between them and the PTP specification.

Mandatory: The TPM must support interrupt driven communication. This requirement applies to discrete TPM devices only.

- **Informative:** TPM vendors can enable interrupt support through the registry settings and can submit the HLK results with Interrupts enabled.

Mandatory: When using TPM_ALG_RSAPSS padding scheme the salt size must be equal to the hash size.

Mandatory: Support for the following.

1. TPMI_AES_KEY_BITS with 256 as an allowed value for AES_KEY_SIZES_BITS
2. Internal use of symmetric cryptography must use a cipher with equivalent strength to TPM_ALG_AES with 256-bit strength.
3. TPM_ALG_SHA_384
4. TPM_ECC_NIST_P384 for TPM_ALG_ECC
5. TPM_ALG_RSA with a Key Size of 3072 bits with AES_KEY_SIZES_BITS of 256

Optional: Support for the following. Effective with the First Windows Release in 2026, these become **Mandatory:**

1. TPM_ECC_NIST_P521 for TPM_ALG_ECC
2. TPM_ALG_SHA_512

Informative: The TCG TPM Library Specification mandates that the hashing algorithm with the highest security strength is chosen for integrity protection. See Section 30.3.2 Context Integrity Protection in Part 1 of the TPM Library Specification for TPM 2.0 revision 1.38.

Optional: Effective with the First Windows Release in 2025 support for the following becomes Optional. If these commands are implemented, they still must meet all applicable performance requirements.

1. TPM_ECC_NIST_P256
2. TPMI_AES_KEY_BITS with 128 as an allowed value for AES_KEY_SIZES_BITS
3. TPM_ALG_SHA_256
4. TPM_ALG_SHA1

Recommended: Support for enhanced TPM Firmware Updates

1. Internal cryptographic validation of TPM firmware updates.
2. A robust recovery mechanism to ensure that any failure during firmware update does not render the TPM inoperable. One such implementation would be sufficient NV space for side-by-side firmware versions with full functionality fallback in the case of a failure during the update process.

Mandatory: The TPM has enough NV space to meet the requirements listed below plus indices used for EK Cert storage as defined by the TCG PC Client Platform TPM Profile (PTP) specification.

- The TPM should support creation of at least 50 NV Indices of at least 32 bytes each for use by the OS. Additional NV indices may be required by the platform.
 - If the TPM supports TPM_ALG_RSA with key size of 3072 or larger and AES_KEY_SIZES_BITS of 256, then it is acceptable that the TPM supports the creation of at least 40 indices of at least 32 bytes each for use by the OS.
- The TPM should not arbitrarily restrict the size of indices if there is sufficient NV space to create them. The maximum size is determined by the value returned by the property TPM_PT_NV_INDEX_MAX.
- The TPM should not arbitrarily restrict the number of indices if there is sufficient NV space to create them.
 - Alternately if the TPM can create at least 128 indices of 32 bytes each, it is acceptable to limit the number of indices created, even if there would be additional NV Space available.
- The TPM should not arbitrarily restrict what defined NV indices may be used for. For example, there should be no arbitrary limit on the number of NV indices which may be used as counters. The number of counters with TPM_NV_ORDERLY set is restricted by the amount of RAM available in the TPM and may be less than the total number of NV counters available.
- The TPM should support the NV indices which are required by the TCG PC Client Platform TPM Profile (PTP) specification for specific purposes.

Recommended: If the TPM device is FIPS 140-3 level 2 or higher certified, please contact Microsoft at tpmkit@microsoft.com for guidance on support of FIPS 140-3 in Windows.

Device.TrustedPlatformModule.TPM20.MinimumPerformance

All TPM 2.0 Devices must meet a minimum performance bar.

Applies to	Windows 11 Client x64 Windows 11 Client ARM64

Mandatory: The TPM shall Comply with the following performance requirements. These are measured with a tool which issues commands from the OS context. Times specified as a mean refer to the mean of 100 or more operations

completed in direct succession. For operations on keys other than import, the operations may be assumed to be being completed on imported (internal) keys.

Mandatory: TPM hardware in general must provide reasonable performance for an interactive user experience. Microsoft reserves the right to modify any prospective requirements, including adding additional test cases, as it deems appropriate.

Mandatory: If any command is listed below with a performance requirement, that command must be implemented.

Mandatory: performance requirements below apply across all localities

In the below table, if unspecified, all TPM_ALG_RSA keys are 2048-bit keys and TPM_ALG_ECC keys are of the TPM_ECC_NIST_256 curve.

Command	Arguments	Mean Requirement (ms)
TPM2_Create	TPM_ALG_RSA with Bits=2048	15,000
	TPM_ALG_ECC with TPM_ECC_NIST_P256, <i>including FIPS pairwise consistency checking when needed</i>	200
	TPM_ALG_ECC with TPM_ECC_NIST_P384, <i>including FIPS pairwise consistency checking when needed</i>	200
	TPM_ALG_KEYEDHASH	100
	32 Bytes	100
TPM2_CreatePrimary	TPM_ALG_RSA with Bits=2048 with AES 128	15,000
	TPM_ALG_RSA with Bits=2048 with AES 256	15,000
	TPM_ALG_RSA with Bits=3072 with AES 256	60000
	TPM_ALG_ECC, TPM_ECC_NIST_P256	200
	TPM_ALG_ECC, TPM_ECC_NIST_P384	200
	TPM_ALG_KEYEDHASH	100
	32 Bytes	250
TPM2_Load	TPM_ALG_ECC, TPM_ECC_NIST_P256 Parent = TPM_ALG_ECC, TPM_ECC_NIST_P256	50
	TPM_ALG_ECC, TPM_ECC_NIST_P384 Parent = TPM_ALG_ECC, TPM_ECC_NIST_P384	80

Command	Arguments	Mean Requirement (ms)
	TPM_ALG_KEYEDHASH Parent = TPM_ALG_RSA, 2048	50
TPM2_Sign	TPM_ALG_ECC, TPM_ECC_NIST_P256 Parent = TPM_ALG_ECC, TPM_ECC_NIST_P256	80
	TPM_ALG_ECC, TPM_ECC_NIST_P384 Parent = TPM_ALG_ECC, TPM_ECC_NIST_P384	120
TPM2_StartAuthSession, TPM2_Load, TPM2_Sign, TPM2_FlushContext, sequentially, combined	TPM_ALG_RSA, 2048 Parent = TPM_ALG_RSA, 2048	300
TPM2_VerifySignature	TPM_ALG_RSA, 2048	50
	TPM_ALG_ECC, TPM_ECC_NIST_P256	100
	TPM_ALG_ECC, TPM_ECC_NIST_P384	100
TPM2_ECDH_ZGen	TPM_ALG_ECC, TPM_ECC_NIST_P256	200
TPM2_StartAuthSession, TPM2_Import, TPM2_Load, TPM2_FlushContext, sequentially, combined	TPM_ALG_RSA, 2048 Parent = TPM_ALG_RSA, 2048 InSymSeed contains asymmetrically encrypted seed using RSA algorithm	700
TPM2_Import	TPM_ALG_RSA, 2048 Parent = TPM_ALG_RSA, 2048	500
	TPM_ALG_ECC, TPM_ECC_NIST_P256 Parent = TPM_ALG_ECC, TPM_ECC_NIST_P256 InSymSeed contains asymmetrically encrypted seed using ECDH algorithm	250
	TPM_ALG_ECC, TPM_ECC_NIST_P384 Parent = TPM_ALG_ECC, TPM_ECC_NIST_P384 InSymSeed contains asymmetrically encrypted seed using ECDH algorithm	300
TPM2_RSA_Encrypt	2048-bit RSA key, 256 bytes	50
TPM2_StartAuthSession, TPM2_Load, TPM2_RSA_Decrypt, TPM2_FlushContext, sequentially, combined	TPM_ALG_RSA, 2048 Parent = TPM_ALG_RSA, 2048, 256 bytes	300
TPM2_HMAC	HMAC Operation	30
TPM2_GetCapability, TPM2_StartAuthSession, TPM2_HMAC		100
TPM2_PolicySigned	Ecdsa NistP256	100
	Sapss RSA 2048	50
	Sassa RSA 2048	50

Command	Arguments	Mean Requirement (ms)
	TPM_ALG_ECC, TPM_ECC_NIST_P384	100
TPM2_PolicyTicket		30
TPM2_Unseal	32 Bytes	30
	With encrypt session	120
TPM2_PCR_Extend	32 Bytes	10
	48 Bytes	10
TPM2_StartAuthSession	Bind is not TPM_RH_NULL	30
	Bind is TPM_RH_NULL	15
TPM2_LoadExternal	TPM_ALG_RSA, 2048 Public key portion	450
TPM2_ReadPublic	TPM_ALG_RSA, 2048	20
	TPM_ALG_ECC, TPM_ECC_NIST_P256	20
TPM2_ActivateCredential	TPM_ALG_RSA, 2048	600
TPM2_ObjectChangeAuth	TPM_ALG_RSA, 2048	200
TPM2_Duplicate	TPM_ALG_RSA, 2048	100
TPM2_ECC_Parameters		20
TPM2_GetRandom		20
TPM2_StirRandom		75
TPM2_Certify	TPM_ALG_RSA, 2048	350
TPM2_CertifyCreation	TPM_ALG_RSA, 2048	400
TPM2_Quote	TPM_ALG_RSA, 2048	350
TPM2_NV_Certify	TPM_ALG_RSA, 2048	400
TPM2_PCR_Event		50
TPM2_PCR_Read		20
TPM2_PolicySecret		50
TPM2_PolicyOr		50
TPM2_PolicyPCR		50
TPM2_PolicyLocality		50
TPM2_PolicyNV		50
TPM2_PolicyCounterTimer		50
TPM2_PolicyCommandCode		50
TPM2_PolicyCpHash		50
TPM2_PolicyNameHash		50
TPM2_PolicyDuplicationSelect		50
TPM2_PolicyAuthorize		50
TPM2_PolicyAuthValue		50
TPM2_PolicyPassword		50
TPM2_PolicyGetDigest		50

Command	Arguments	Mean Requirement (ms)
TPM2_PolicyNvWritten		50
TPM2_HierarchyChangeAuth		50
TPM2_DictionaryAttackLockReset		50
TPM2_DictionaryAttackParameters		50
TPM2_FlushContext		20
TPM2_EvictControl		230
TPM2_ReadClock		50
TPM2_GetCapability		50
TPM2_NV_DefineSpace	32 Bytes	100
TPM2_NV_UndefineSpace	32 Bytes	100
TPM2_NV_ReadPublic		50
TPM2_NV_ReadPublic, TPM2_StartAuthSession, TPM2_NV_Read		130
TPM2_NV_Increment		50
TPM2_NV_Extend	32 Bytes, Subsequent Writes into Previously Defined Index	50
TPM2_NV_SetBits	First Write into Newly Defined Index	150
TPM2_NV_Read	32 Bytes	50
TPM2_NV_ChangeAuth		50
TPM2_NV_Write	32 Bytes, First Write into Newly Defined Index	150
TPM2_NV_Write	32 Bytes, Subsequent Writes into Previously Defined Index	50
TPM2_ContextSave		50
TPM2_ContextLoad		50

1. The performance requirements below come into effect with first major Windows release of 2024. When in conflict with performance requirements above, they are superseding.

Command	Arguments	Mean Requirement (ms)
TPM2_PCR_Read, TPM2_StartAuthSession, TPM2_Unseal, TPM2_FlushContext, TPM2_PCR_Extend	Unseal without encrypt session, PCR with SHA256 PCR	240

Device.TrustedPlatformModule.TPM20.EKCert

Systems shipping with TPM 2.0 must have appropriate associated EK Certificates available

Applies to	Windows 11 Client x64

Description**Mandatory**

1. All TPMs must contain a full Endorsement Key (EK) certificate stored in the TPM NV RAM as described in the "TCG EK Credential Profile for TPM Family 2.0, Level 0, Version 2.1, Revision 13" or have an associated certificate available to obtain from a public facing server via a documented process.
2. The Issuing Authorities for the EK Certificates or EK Cert intermediates on the Device must be provided to Microsoft before a Device may be certified.

Informative: The CA key used to sign an RSA EK certificate may use the RSA algorithm also, where it SHALL have an equal or higher security strength than the EK.

Mandatory: TPM manufacturer must provide an EK Certificate for the TPM_ECC_NIST_P384 curve.

Mandatory: TPM manufacturer must provide an EK Certificate for the RSA-2048 EK, created from the default low range template.

Mandatory: If an EK certificate for the RSA 2048-bit EK is populated in the TPM at shipping it must be populated at the low range NV location that adheres to the EK Credential Profile Specification and must use the template for the low range NV location.

Mandatory: If an EK certificate for ECC P384 curve EK is populated in the TPM at shipping it must be populated at the NV location that adheres to the EK Credential Profile Specification and must use the respective template.

The EK certificate must be capable of being 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 an EK Certificate. The OID used for this purpose is "2.23.133.8.1".

The EK certificate may also be signed using ECDSA. The supported ECC curves for this purpose are NIST 256, NIST 384 and NIST 521.

The EK certificate or on-device EK certificate intermediate CA must contain an AIA extension that contains the URL for the parent issuing CA Certificate in the certificate chain.

The AIA extension must also be present in each non-root cert in the chain with URLs that make the issuing CA certificate (any intermediate CA certs and the root CA cert) – discoverable and retrievable via iterative fetching when starting only with the certificates on the device.