Universal Serial Bus Revision 3.2

USB Command Verifier Compliance Test Specification

Revision 1.22

Date: July 10, 2019

Revision: 1.22

Intellectual Property Disclaimer

Revision History

Revision	Issue Date	Comments
0.01	August 15, 2013	Pre-Initial Version
0.5	January 21, 2014	Assertions done for USB 3.1
0.7	April 2, 2015	Test Specifications updated
0.72	December 1, 2015	TD 9.1 updated
0.73	January 29, 2016	TD 9.22 updated
0.74	March 18, 2016	TD 9.20 updated
1.00	February 14, 2017	Official Release Version
1.01	December 18, 2017	TD 9.15 updated
1.21		Updated for USB 3.2
1.22	February 28, 2019	Clean up assertions references for TD 9.14, TD 9.15, TD 9.20, TD 9.22, TD 9.24, TD 9.25

Significant Contributors:

Please send comments via electronic mail to: techadmin@usb.org or ssusbcompliance@usb.org

1 Introduction

2 Assertions to verify that USB devices are compliant with the USB 3.1 Specification.

General Requirements:

- All Reserved fields shall be set to 0.

Assertion #	Assertion Description	Test #
Subsection reference: 4.4.7.1 Interrupt Transfer Packet Size		
4.4.7.1#1	An Enhanced SuperSpeed default interface cannot contain an Interrupt endpoint with a maximum packet size greater than 64.	9.4
Subsection reference: 4.4.7.2 Interrupt Transfer Bandwidth Requirements		
4.4.7.2#1	An Enhanced SuperSpeed Interrupt endpoint can move up to 3 packets per service interval (bMaxBurst <= 2).	9.6
Subsection reference: 4.4.8.1 Isochronous Transfer Packet Size		
4.4.8.1#1	An Enhanced SuperSpeed default interface cannot contain an Isochronous endpoint with a maximum packet size greater than 0.	9.4
Subsection reference: 8.4.2 Set Link Function		
8.4.2#1	Upon receipt of a LMP with Force_LinkPM_Accept bit set, a device must accept all LGO_U1 and LGO_U2 requests.	
Subsection reference: 8.5.6.7 Sublink Speed Device Notification		
8.5.6.7#1	Sublink Speed Device Notification TPs shall be generated by Enhanced SuperSpeed devices operating in SuperSpeedPlus mode upon entering the Address USB Device State.	9.29
8.5.6.7#2	A device shall inform the host that a Device Notification TP shall follow a SET_ADDRESS request by setting the TPF flag in the Status stage ACK TP.	9.29
8.5.6.7#3	The Rx Sublink Device Notification TP shall be in the next TP transmitted by a device after setting the TPF flag in a Status stage ACK TP.	9.29
8.5.6.7#4	The device shall inform the host that a second Device Notification TP shall follow the Rx Sublink Speed Device Notification TP.	9.29
8.5.6.7#5	The Tx Sublink Speed Device Notification TP shall be the next TP transmitted by a device after setting the TPF flag in the Rx Sublink Speed Device Notification TP.	9.29

Assertion #	Assertion Description	Test #
8.5.6.7#6	Asymmetric Lane Types may only be reported by SSIC devices.	9.29
8.5.6.7#7	A non-SSIC link (Symmetric) shall report the same Lane Speed for both Rx and Tx.	9.29
8.5.6.7#8	Enhanced SuperSpeed devices shall only support Symmetric links.	9.29
8.5.6.7#9	Link Protocol shall be 1 for SuperSpeedPlus devices.	9.29
8.5.6.7#10	A Symmetric link is one that has the same number of lanes for both Rx and Tx Sublinks.	9.29
Subsection reference: 9.1.1 USB Device States		
9.1.1#1	Devices must have a corresponding configuration value for a valid configuration index.	Test Initialization
9.1.1#2	Devices must support being set to Addressed/Configured state.	Test Initialization
9.1.1.3#1	Enhanced SuperSpeed capable devices must operate at SuperSpeed when connected to a Enhanced SuperSpeed capable host.	Test Initialization
9.1.1.3#2	Enhanced SuperSpeed devices must support operation at 5Gb/s.	9.7
9.1.1.3#3	Enhanced SuperSpeed devices must support operation at SuperSpeed and at one of the supported USB 2.0 speeds.	9.7
9.1.1.3#4	USB 3.1 compliant devices must reset successfully at one of the supported USB 2.0 speeds when in an USB 2.0 only electrical environment.	Interop Test Procedure
9.1.1.3#5	A hub or peripheral device shall transition from the Default state to the Powered::Far-end Receiver Termination substate if the hub or peripheral device receives a Warm Reset or the hub or device initiates a speed change or a speed change is initiated on the hub or peripheral device’s upstream facing port.	
9.1.1.4#1	Devices must use the default address when initially powered or reset.	Test Initialization
9.1.1.4#2	A hub or peripheral device shall transition from the Addressed state to the Powered::Far-end Receiver Termination substate if the hub or peripheral device receives a Warm Reset or the hub or device initiates a speed change or a speed change is initiated on the hub or peripheral device’s upstream facing port.	
9.1.1.5#1	Devices must default into the fully functional D0 State on initial entry into the Configured state	Test Initialization

Assertion #	Assertion Description	Test #
9.1.1.5#2	Configuring a device or changing an alternate setting shall reset the status and configuration values for all affected interfaces to their default value	
9.1.1.5#3	A hub or peripheral device shall transition from the Configured state to the Powered::Far-end Receiver Termination substate if the hub or peripheral device receives a Warm Reset or the hub or device initiates a speed change or a speed change is initiated on the hub or peripheral device's upstream facing port.	
9.1.1.6#1	Devices must automatically enter the suspended state when they observe that the upstream link is being driven to the U3 state.	9.14
9.1.1.6#2	Device shall exit suspend mode when it observes up-stream signaling	9.14
9.1.1.6#3	If a device is capable of remote wake, the device shall support the ability of the host to enable and disable this capability	9.15
9.1.1.6#4	When a device is reset, remote wakeup shall be disabled	9.15
Subsection reference: 9.2 Generic Device Operations		
9.2.3#1	Device must be configured before any of its functions can be used	
9.2.3#2	Default setting when a device is initially configured is alternate setting zero	9.13
9.2.4#1	Once an alternate setting is selected a device endpoint shall use only one data transfer method until a different alternate setting is selected	
9.2.5.1#1	Devices must limit the power they consume from Vbus to one unit load or less until configured	Interop Test Procedure
9.2.5.1#2	An Enhanced SuperSpeed device operating in Enhanced SuperSpeed mode shall draw no more than 6 unit loads from Vbus (once it has been configured)	Interop Test Procedure
9.2.5.1#3	An Enhanced SuperSpeed device operating in a USB 2.0 environment shall draw no more than 500mA from Vbus (once it has been configured)	Interop Test Procedure
9.2.5.1#4	Suspended devices, whether configured or not, must limit their bus power consumption as to the suspend mode power requirements in the USB 2.0 specification.	Interop Test Procedure
9.2.5.2#1	Devices must send a Function Wake Notification after driving resume signaling	9.15

Assertion #	Assertion Description	Test #
9.2.5.2#2	If the device has not been accessed for longer than tNotification since sending the last Function Wake Notification, the device must send the Function Wake Notification again until it has been accessed.	9.15
9.2.5.2#3	Devices maintain device address when resuming from the suspend state	9.14
9.2.5.2#4	Devices maintain device configuration when resuming from the suspend state	9.14
9.2.5.2#5	Devices maintain function suspend and function remote wake enable state when resuming from the suspend state	9.15
9.2.5.4#1	If the link for a device that is exiting function suspend is in a non-U0 state, then the device shall transition the link to U0 before sending a remote wake message.	9.15
9.2.5.4#2	When all functions within a device are in function suspend and the PORT_U2_TIMEOUT field is programmed to 0xFF, the device shall initiate U2 after 10ms of link inactivity.	9.14
9.2.5.4#3	If a remote wake event occurs in multiple functions, each function shall send a Function Wake	Interop Test Procedure
9.2.6.1#1	Devices must process any command in no more than 5 seconds.	Test Initialization
9.2.6.2#1	After a port reset or resume device attached to port shall immediately respond to data transfers	9.14, 9.16
9.2.6.3#1	Devices must be able to complete processing of the SetAddress() request after reset or resume and must successfully complete the Status stage of the request within 50 ms.	Test Initialization
9.2.6.3#2	Devices must not respond to transactions sent to the old address after successful completion of the Status stage	
9.2.6.4#1	Devices must be able to complete standard device requests that require no Data stage and must be able to successfully complete the Status stage of the request within 50 ms of receipt of the request.	Test Initialization
9.2.6.4#2	For standard device requests that require a data stage transfer to the host, the device must return the first data packet to the host within 500 ms of receipt of the request. For subsequent data packets, if any, the device must return them within 500 ms of successful completion of the transmission of the previous packet. The device must then successfully complete the status stage within 50 ms after returning the last data packet.	9.19
9.2.6.6#1	Devices capable of operation at SuperSpeed shall be fully functional at one of the USB 2.0 defined speeds.	Interop Test Procedure

Assertion #	**Assertion Description**	**Test #**
Subsection reference: 9.3 USB Device Requests		
9.3#1	In response to a read control transfer, devices must return up to the length of data as specified in the wLength field of the Setup request	Test Initialization
Subsection reference: 9.4 Standard Device Requests		
9.4#1	Devices must respond by returning a STALL Transaction Packet in the Data or status stage of the request in response to an unsupported or invalid request.	9.10, 9.11
9.4#2	Device shall not set the Halt feature on the Control pipe upon receipt of an unsupported or invalid request	9.10, 9.11
Subsection reference: 9.4.1 Clear Feature		
9.4.1#1	Devices must respond with a Request Error to any ClearFeature() request that references a feature that cannot be cleared, that does not exist, or that references an interface or an endpoint that does not exist	9.11
9.4.1#2	Devices must respond to a ClearFeature() request in the Address or Configured state.	9.9
9.4.1#3	Device must clear U1_Enable feature in response to a ClearFeature(U1_Enable) only if it is in the Configured state.	9.22
9.4.1#4	Device must clear U2_Enable feature in response to a ClearFeature(U2_Enable) only if it is in the Configured state.	9.22
9.4.1#5	Device must clear LTM_Enable feature in response to a ClearFeature(LTM_Enable) only if it is in the Configured state.	9.20
9.4.1#6	Device must clear LDM_Enable feature in response to a ClearFeature(LDM_Enable) only if it is in the Configured state.	9.22
9.4.1#7	Devices must disable/clear the feature in the wValue field in response to the ClearFeature() request.	9.9
Subsection reference: 9.4.2 Get Configuration		
9.4.2#1	Devices should return a zero in response to the GetConfiguration() request in the Address state.	9.13
9.4.2#2	Devices should return the non-zero bConfigurationValue of the current configuration for the GetConfiguration() request in the Configured state.	Test Initialization, 9.13
Subsection reference: 9.4.3 Get Descriptor		

Assertion #	Assertion Description	Test #
9.4.3#1	Devices must respond with a Request Error to GetDescriptor() requests that specify an unsupported descriptor type.	9.10
9.4.3#2	Devices must support a valid GetDescriptor(Device) request.	9.1, 9.10, 9.11, 9.14
9.4.3#3	Devices must support a valid GetDescriptor(DeviceQualifier) request when operating in one of the USB 2.0 defined speeds	Chap 9 Tests for 2.0 devices
9.4.3#4	Devices must return the endpoint companion descriptors in addition to configuration descriptor, all interface descriptors and endpoint descriptors for all interfaces in response to GetDescriptor(Configuration) in a single request when operating in SuperSpeed mode.	9.5, 9.6, 9.9
9.4.3#5	Devices must support a valid GetDescriptor(String) request.	Test Initialization, 9.8
9.4.3#6	Devices must support a valid GetDescriptor(Configuration) request.	Test Initialization, 9.2, 9.3, 9.4, 9.5, 9.6, 9.14, 9.15
9.4.3#7	High Speed Capable devices must support a valid GetDescriptor(OtherSpeedConfiguration) request.	Chap 9 Tests for 2.0 devices
9.4.3#9	Devices must support a valid GetDescriptor(BOS) request.	9.7
9.4.3#10	An Enhanced SuperSpeed device shall not report device_qualifier descriptor when operating in SuperSpeed mode	Test Initialization, 9.2
9.4.3#11	An Enhanced SuperSpeed device shall not report other_speed_configuration descriptor when operating in SuperSpeed mode	Test Initialization, 9.2
Subsection reference: 9.4.4 Get Interface		
9.4.4#1	Device must support the GetInterface() request if it has alternate settings for that interface.	9.4
9.4.4#2	A successful GetInterface() request must return the alternate setting set by a prior call to SetInterface.	9.4
9.4.4#3	Devices must return a Request Error in response to the GetInterface() request in the Address State.	
9.4.4#4	Devices must respond with a Request Error for GetInterface() request, if the Interface specified does not exist.	
9.4.4#5	An Interface must have at least one setting with an Alternate Setting set to zero.	9.4
Subsection reference: 9.4.5 Get Status		

Assertion #	**Assertion Description**	**Test #**
9.4.5#1	Devices must return the status of the recipient specified in the Recipient bits of the bmRequestType in response to the GetStatus() request. (Fig 9-5, 9-6, 9-7)	9.2, 9.9, 9.15, 9.20, 9.22
9.4.5#2	Devices must respond with a Request Error in response to a GetStatus() request, if an interface or endpoint is specified which does not exist, in the Configured State.	
9.4.5#3	Devices must respond with a Request Error in response to the GetStatus() request, if an interface or endpoint other than the Default Control Pipe is specified, in the Address State.	
9.4.5#4	For a self-powered device, Bit D0 of status word returned in response to a GetStatus(Device) request, must be set to a one.	Interop Test Procedure
9.4.5#5	For a bus-powered device, Bit D0 of status word returned in response to a GetStatus(Device) request, must be set to a zero	Interop Test Procedure
9.4.5#6	Enhanced SuperSpeed Devices must set Bit D1 (REMOTE_WAKEUP) of the status word, returned in response to GetStatus(Device) to zero	9.14
9.4.5#7	After a successful ClearFeature(U1_Enable) Bit D2 of the status word returned in response to a GetStatus(STANDARD_STATUS) request must be set to a zero	9.22, 9.24, .9.25
9.4.5#8	After a successful SetFeature(U1_Enable) Bit D2 of the status word returned in response to a GetStatus(STANDARD_STATUS) request must be set to a one.	9.22, 9.24, 9.25
9.4.5#9	After a successful ClearFeature(U2_Enable) Bit D3 of the status word returned in response to a GetStatus(STANDARD_STATUS) request must be set to a zero	9.22, 9.24, 9.25
9.4.5#10	After a successful SetFeature(U2_Enable) Bit D3 of the status word returned in response to a GetStatus(STANDARD_STATUS) request must be set to a one.	9.22
9.4.5#11	After a successful SetFeature(LTM_ENABLE) Bit D4 of the status word returned in response to a GetStatus(STANDARD_STATUS) request must be set to a one.	9.20, 9.22
9.4.5#12	After a successful ClearFeature(LTM_ENABLE) Bit D4 of the status word returned in response to a GetStatus(STANDARD_STATUS) request must be set to a zero.	9.22

Assertion #	Assertion Description	Test #
9.4.5#13	In response to a GetStatus(STANDARD_STATUS) request to the first interface in a function, Bit 0 of the status word returned must be set to a one if the function supports function remote wake up.	9.15
9.4.5#14	After a successful ClearFeature(FUNCTION_ SUSPEND) Bit 1 of the status word returned in response to a GetStatus(STANDARD_STATUS) request to the first interface in a function must be set to a zero	9.14
9.4.5#15	After a successful SetFeature(FUNCTION_SUSPEND) Bit 1 of the status word returned in response to a GetStatus(STANDARD_STATUS) request to the first interface in a function must be set to a one.	9.14
9.4.5#16	A GetStatus(STANDARD_STATUS) request to any interface other than the first interface in a function must return all zeros in the status word.	
9.4.5#17	In response to a GetStatus(STANDARD_STATUS) request to an Interrupt or Bulk endpoint, if the SetFeature(ENDPOINT_HALT) to the Interrupt or Bulk endpoint is completed successfully, then Bit 0 of the status word returned must be set to a one.	9.9
9.4.5#18	After the successful completion of a ClearFeature(ENDPOINT_HALT), Bit 0 of status word returned in response to a GetStatus(STANDARD_STATUS) request to an Interrupt or Bulk endpoint must be set to a zero.	9.9
9.4.5#19	The *Self Powered* field in the device must not be changed by the SetFeature() or ClearFeature() requests.	
9.4.5#20	In response to a GetStatus(STANDARD_STATUS) request to a function, bit 0 should be 1 if the function supports remote wake up.	9.15
9.4.5#21	Bit D2 (U1_ENABLE) of the status word returned in response to a GetStatus(STANDARD_STATUS) request to a device must be reset to a zero when the device is reset.	9.22
9.4.5#22	Bit D3 (U2_ENABLE) of the status word returned in response to a GetStatus(STANDARD_STATUS) request to a device must be reset to a zero when the device is reset.	9.22
9.4.5#23	Bit D4 (LTM_ENABLE) of the status word returned in response to a GetStatus(STANDARD_STATUS) request to a device must be reset to a zero when the device is reset.	9.22

Assertion #	Assertion Description	Test #
9.4.5#24	Bit D1 (Function Remote Wakeup) of the status word returned in response to a GetStatus() request to the first interface in a function must be reset to a zero when the function is reset.	9.15
9.4.5#25	Devices must support a GetStatus() Standard Request.	9.2, 9.9, 9.15
9.4.5#26	An Enhanced SuperSpeed device operating in USB 2.0 mode that supports remote wakeup in a specific configuration must not fail a valid SetFeature(DEVICE_REMOTE_WAKEUP) command.	Chap 9 Tests for 2.0 devices
9.4.5#27	An Enhanced SuperSpeed device operating in USB 2.0 mode that supports remote wakeup in a specific configuration must not fail a valid ClearFeature(DEVICE_REMOTE_WAKEUP) command.	Chap 9 Tests for 2.0 devices
9.4.5#28	After a successful ClearFeature(DEVICE_REMOTE_WAKEUP) Bit 1 of the status word returned in response to a GetStatus() request must be set to a zero in a SuperSpeed device operating in USB 2.0 mode.	Chap 9 Tests for 2.0 devices
9.4.5#29	After a successful SetFeature(DEVICE_REMOTE_WAKEUP) Bit 1 of the status word returned in response to a GetStatus() request must be set to a one in an Enhanced SuperSpeed device operating in USB 2.0 mode.	Chap 9 Tests for 2.0 devices
9.4.5#30	Devices with remote wakeup disabled must not initiate a remote wakeup.	9.15
9.4.5#31	Device with remote wakeup enabled must be able to initiate a remote wakeup on its suspended parent port.	9.15
9.4.5#32	Enhanced SuperSpeed devices with function remote wakeup disabled must not initiate a function remote wakeup.	9.15
9.4.5#33	Enhanced SuperSpeed devices with function remote wakeup enabled must be able to initiate a function remote wakeup.	9.15
9.4.5#34	A suspended device must resume normal operation when resume signaling is seen on its upstream port	9.14
9.4.5#35	Bulk and Interrupt endpoints must support ENDPOINT_HALT.	9.9
9.4.5#36	Immediately after the successful completion of a SetConfiguration() or SetInterface(), Bit 0 of the status word (HALT) returned in response to a GetStatus() request to an Interrupt or Bulk endpoint must be set to a zero.	

Assertion #	Assertion Description	Test #
9.4.5#37	If the low order byte of wValue is 0, then the device shall return a 2-byte standard status register.	9.22
9.4.5#38	If the low order byte of wValue is 1, then the device shall return a 4-byte PTM status register.	9.22
9.4.5#39	If a device supports PTM, then after the successful completion of a SetFeature(LDM_ENABLE), Bit 0 of the status field returned in response to a GetStatus(PTM_STATUS) request to the device must be set to one.	9.22
9.4.5#40	If a device supports PTM, then after the successful completion of a ClearFeature(LDM_ENABLE), Bit 0 of the status field returned in response to a GetStatus(PTM_STATUS) request to the device must be set to zero.	9.22
9.4.5#41	If a device supports PTM, then bit 0 of the status field returned in response to a GetStatus(PTM_STATUS) shall be set to one after the device is reset.	9.22
9.4.5#42	If a device supports PTM, then after the successful completion of a ClearFeature(LDM_ENABLE), Bit 1 of status field returned in response to a GetStatus(PTM_STATUS) request to the device must be set to zero.	9.22
9.4.5#43	If a device supports PTM, then the entire status field returned in response to a GetStatus(PTM_STATUS) shall be set to zero if bit 0 of the status field is set to zero.	9.22
Subsection reference: 9.4.6 Set Address		
9.4.6#1	The SetAddress() request must set the device address as specified for all future device accesses.	Test Initialization
9.4.6#2	In the Address state, for the SetAddress() request, if the address specified is zero the device must enter the Default state.	
9.4.6#3	In the Address state, for the SetAddress() request, if the address specified is not zero, the device must remain in the Address state but must use the newly specified address.	Test Initialization
9.4.6#4	In the Default state, for the SetAddress() request, if the address specified is non-zero, then the device must enter the Address State.	Test Initialization
9.4.6#5	In the Default state, for the SetAddress() request, if the address specified is zero, then the device must remain in the Default state.	

Assertion #	Assertion Description	Test #
9.4.6#6	Device shall not change its device address until after the Status stage of a SetAddress() request completes successfully	Test Initialization
Subsection reference: 9.4.7 Set Configuration		
9.4.7#1	Devices must support a valid SetConfiguration() request	Test Initialization, 9.13
9.4.7#2	In the Address state in response to the SetConfiguration() request, the device must remain in the Address state, if the specified configuration is zero.	9.13
9.4.7#3	In the Address state or the Configured State, the device must respond with a Request Error in response to the SetConfiguration() request, if the configuration value does not match any value from a configuration descriptor and is non-zero.	9.13
9.4.7#4	In the Configured state in response to the SetConfiguration() request, the device must enter the Address state, if the specified configuration is zero.	Test Initialization, 9.13
Subsection reference: 9.4.8 Set Descriptor		
9.4.8#1	The only allowed values for descriptor type in a SetDescriptor() request are device, configuration and string descriptor	
Subsection reference: 9.4.9 Set Feature		
9.4.9#1	Devices must support a valid SetFeature() request.	9.14, 9.15, 9.20
9.4.9#2	Devices must respond with a STALL to SetFeature() requests that specify an invalid or unsupported feature selector.	9.11
9.4.9#3	In the Address State, for the SetFeature() request, if an interface or an endpoint other than the Default Control Pipe is specified then the device must respond with a Request Error.	
9.4.9#4	In the Address State, for the SetFeature() request, if the device receives a SetFeature(U1/U2 Enable or LTM Enable or FUNCTION_SUSPEND), then the device must respond with a Request Error	9.22
9.4.9#5	Device must not fail a valid SetFeature(U1_Enable) command when in the Configured SuperSpeed state.	9.22
9.4.9#6	Device must not fail a valid SetFeature(U2_Enable) command when in the Configured SuperSpeed state.	9.22, 9.24, 9.25

Assertion #	Assertion Description	Test #
9.4.9#7	Device must not fail a valid SetFeature(LTM_Enable) command when in the Configured SuperSpeed state and if it supports LTM capability.	9.20
9.4.9#8	An interface must not fail a valid SetFeature(FUNCTION_SUSPEND) command (in the Configured state).	9.14, 9.15
9.4.9#9	An interface must respond with a Request error if it receives a SetFeature(FUNCTION_SUSPEND) request in the Address state.	
9.4.9#11	Upon receipt of a SetFeature(FUNCTION_SUSPEND) request where Bit 0 of the wIndex field is zero the function shall transition to normal operation mode	9.14, Implicit in Interop Test Procedure
9.4.9#12	Upon receipt of a SetFeature(FUNCTION_SUSPEND) request where Bit 0 of the wIndex field is one the function shall transition to low power suspend state	9.14, Implicit in Interop Test Procedure
9.4.9#13	Upon receipt of a SetFeature(FUNCTION_SUSPEND) request where Bit 1 of the wIndex field is zero the function shall disable function remote wake.	9.15, Implicit in Interop Test Procedure
9.4.9#14	Upon receipt of a SetFeature(FUNCTION_SUSPEND) request where Bit 1 of the wIndex field is one the function shall enable function remote wake.	9.15, Implicit in Interop Test Procedure
9.4.9#15	A device shall support the LDM_ENABLE feature if it is in the Address or Configured states and supports the PTM capability.	9.22
Subsection reference: 9.4.10 Set Interface		
9.4.10#1	In the Configured state, a device must support the SetInterface() request if it has alternate settings for that interface.	9.4
9.4.10#2	In the Address State, for the SetInterface() request, the device must respond with a Request Error.	
Subsection reference: 9.4.11 Set Isochronous Delay		
9.4.11#1	A device must accept the SetIsochDelay() request and shall not respond with a stall or request error.	9.26
Subsection reference: 9.4.12 Set SEL		
9.4.12#1	In the Addressed and Configured states, a device must accept the SetSEL() request and must not respond with a stall or request error.	9.27
Subsection reference: 9.4.13 Sync Frame		

Assertion #	Assertion Description	Test #
9.4.13#1	For the SynchFrame() request, in the Address State, the device must respond with a Request Error	
9.4.13#2	For the SynchFrame() request, in the Configured State, the device must respond with a Request Error if the specified endpoint does not support this request.	
Subsection reference: 9.6 Standard USB Descriptor Definitions		
Subsection reference: 9.6.1 Device Descriptors		
9.6.1#1	The descriptor returned in response to a GetDescriptor(Device) request must return the value of 0x0320 in the bcdUSB field for Enhanced SuperSpeed devices	9.1
9.6.1#2	The MaxPacketSize of a Control endpoint must always be 09H for devices in EnhancedSuperSpeed mode.	9.1
9.6.1#3	The MaxPacketSize of a Control endpoint must be 0x40 for devices operating at High speed.	Chap 9 Tests for 2.0 devices
9.6.1#4	The MaxPacketSize of a Control endpoint must be one of 0x08/0x10/0x20/0x40 for devices operating at Full speed	Chap 9 Tests for 2.0 devices
9.6.1#5	The MaxPacketSize of a Control endpoint must be 0x08 for devices operating at Low speed.	Chap 9 Tests for 2.0 devices
9.6.1#6	Device descriptors must use only Class codes defined in the USB specification, or allocated and published by the USBIF.	9.1
9.6.1#7	Device descriptors must use only SubClass codes defined in the USB specification, or allocated and published by the USBIF.	9.1
9.6.1#8	The descriptor returned in response to a GetDescriptor(Device) request must have a length of 0x12 (or appropriate length if fewer bytes are requested).	9.1
9.6.1#9	The descriptor returned in response to a GetDescriptor(Device) request must return value of 01 (DEVICE) in the bDescriptorType field.	9.1
9.6.1#10	The descriptor returned in response to a GetDescriptor(Device) request must return the value of 0x02 in the high byte of the bcdUSB field for all USB 2.0 speeds	9.2 Chap 9 Tests for 2.0 devices
9.6.1#11	The descriptor returned in response to a GetDescriptor(Device) request must return the value of 0x10 in the low byte of the bcdUSB field for all USB 2.0 speeds	9.2 Chap 9 Tests for 2.0 devices

Assertion #	Assertion Description	Test #
9.6.1#12	If serial number is implemented then each instance of a product shall have a unique serial number	Untestable
Subsection reference: 9.6.2 Binary Device Object Store (BOS)		
9.6.2#1	All Enhanced SuperSpeed devices must have a BOS descriptor.	9.7
9.6.2#2	An Enhanced SuperSpeed device must support LPM when operating in USB 2.0 mode.	Chap 9 Tests for 2.0 devices
9.6.2#3	BOS descriptor field wTotalLength must accurately reflect the length of the BOS descriptor and all of its sub descriptors.	9.7
9.6.2#4	BOS descriptor field bLength must be 0x05.	9.7
9.6.2#5	BOS descriptor field bDescriptorType must be 15 (BOS)	9.7
9.6.2#7	BOS.bNumDeviceCaps must reflect the number of separate device capability descriptors in the BOS.	9.7
9.6.2.1#1	An Enhanced SuperSpeed device must include the USB 2.0 Extension descriptor	9.7
9.6.2.1#2	The USB 2.0 Extension descriptor returned in response to a GetDescriptor(BOS) request must return value of 16 (DEVICE_CAPABILITY) in the bDescriptorType field.	9.7
9.6.2.1#3	The USB 2.0 Extension descriptor returned in response to a GetDescriptor(BOS) request must return value of 0x02 (USB 2.0 EXTENSION) in the Device Capability Type field.	9.7
9.6.2.1#4	Bit 1 in Attributes field of a USB 2.0 Extension descriptor returned in response to a GetDescriptor(BOS) request must be 1 for Enhanced SuperSpeed devices.	9.7
9.6.2.1#5	Bit 0 and bits 31:2 in Attributes field of a USB 2.0 Extension descriptor returned in response to a GetDescriptor(BOS) request are Reserved and must be set to zero.	9.7
9.6.2.2#1	An Enhanced SuperSpeed device must have a SuperSpeed USB Device Capability descriptor.	9.7
9.6.2.2#2	The SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must return value of 16 (DEVICE_CAPABILITY) in the bDescriptorType field.	9.7

Assertion #	Assertion Description	Test #
9.6.2.2#3	The SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must return value of 0x03 (SUPERSPEED_USB) in the Device Capability Type field.	9.7
9.6.2.2#4	Bit 1 in Attributes field of a SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must be one for devices capable of generating Latency Tolerance Messages.	9.22
9.6.2.2#5	Bit 0 and bits 7:2 in Attributes field of a SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request are Reserved and must be set to zero.	9.7
9.6.2.2#6	Bit 0 in Speeds Supported field of a SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must be one if the device supports operation at Low-speed USB.	Implicit in Interop Test Procedure, Chap 9 Tests for 2.0 Devices
9.6.2.2#7	Bit 1 in Speeds supported field of a SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must be one if the device supports operation at Full-speed USB.	Implicit in Interop Test Procedure, Chap 9 Tests for 2.0 Devices
9.6.2.2#8	Bit 2 in Speeds supported field of a SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must be one if the device supports operation at High-speed USB.	Implicit in Interop Test Procedure, Chap 9 Tests for 2.0 Devices
9.6.2.2#9	Bit 3 in Speeds supported field of a SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must be one for all Enhanced SuperSpeed devices.	9.7
9.6.2.2#10	Bits 15:4 in Speeds supported field of a SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request are Reserved and must be set to zero.	9.7
9.6.2.2#11	The SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must return the lowest speed at which all the functionality supported by the device available to the user(as specified in the Speeds Supported field) in the Functionality Support field	9.7
9.6.2.2#12	The SuperSpeed Device Capability descriptor returned in response to a GetDescriptor(BOS) request must return value of 0xA in the bLength field.	9.7

Assertion #	Assertion Description	Test #
9.6.2.2#13	The SuperSpeed Device Capability descriptor returned in response to a GetDescriptor(BOS) request must not contain a value of 0x4 or higher in the bFunctionalitySupport field	9.7
9.6.2.2#14	The SuperSpeed Device Capability descriptor returned in response to a GetDescriptor(BOS) request must not contain a value of 0x0B or higher in the bU1DevExitLat field	9.7
9.6.2.2#15	The SuperSpeed Device Capability descriptor returned in response to a GetDescriptor(BOS) request must not contain a value of 0x0800 or higher in the wU2DevExitLat field	9.7
9.6.2.2#16	Devices with a BCD less than or equal to 0x0200 must not have a BOS descriptor	Chap 9 Tests for 2.0 Devices
9.6.2.2#17	The USB 2.0 Extension Capability descriptor returned in response to a GetDescriptor(BOS) request must return value of 0x7 in the bLength field.	9.7
9.6.2.3#1	If device is of class hub then device shall implement a Container ID descriptor	9.7
9.6.2.3#2	If a device implements a Container ID descriptor then it shall be provided when operating in any mode	
9.6.2.3#3	If a device implements a Container ID descriptor then the length field of a container ID descriptor shall be 14H	9.7
9.6.2.3#4	If a device implements a Container ID descriptor then the Container ID descriptor returned in response to a GetDescriptor(BOS) request must return value of 16 (DEVICE_CAPABILITY) in the bDescriptorType field.	9.7
9.6.2.3#5	If a device implements a Container ID descriptor then the reserved field must be set to zero	9.7
9.6.2.3#6	If a Container ID is implemented then each instance of the device shall have a unique Container ID	9.7
9.6.2.3#7	If a device implements a Container ID descriptor then the Container ID descriptor returned in response to a GetDescriptor(BOS) request must return value of 0x04 (CONTAINER_ID) in the DevCapabilityType field	9.7
9.6.2.4#1	If a device implements a Platform capability descriptor, then the length field must be greater than or equal to 20.	9.7
9.6.2.4#2	If a device implements a Platform capability descriptor, then the reserved field must be set to zero.	9.7

Assertion #	Assertion Description	Test #
9.6.2.4#3	If a device implements a Platform descriptor then the Platform descriptor returned in response to a GetDescriptor(BOS) request must return value of 16 (DEVICE_CAPABILITY) in the bDescriptorType field.	
9.6.2.4#4	If a device implements a Platform descriptor then the Platform descriptor returned in response to a GetDescriptor(BOS) request must return value of 0x05 (PLATFORM) in the DevCapabilityType field	9.7
9.6.2.5#1	A SuperSpeed Plus capability descriptor's must have a length of 12 + ((bmAttributes.SSAC + 1) * 4).	9.7
9.6.2.5#2	A SuperSpeed Plus capability descriptor's reserved fields must be set to zero.	9.7
9.6.2.5#3	Bits 31:9 of the bmAttributes field must be set to zero.	9.7
9.6.2.5#4	Bits 3:0 of wFunctionalitySupport must be equal to the SSID of one of the bmSublinkSpeedAttr fields of this descriptor.	9.7
9.6.2.5#5	Bits 7:4 of wFunctionalitySupport must be set to zero.	9.7
9.6.2.5#6	Bits 11:8 of wFunctionalitySupport must be set to one.	9.7
9.6.2.5#7	Bits 15:12 of wFunctionalitySupport must be set to one.	9.7
9.6.2.5#8	For each SSID, there must be two bmSublinkSpeedAttr fields, an Rx (bit 7 = 0) followed by a Tx (bit 7 = 1).	9.7
9.6.2.5#9	Each Sublink must be symmetric (bit 6 = 0).	9.7
9.6.2.5#10	Bits 13:8 must be set to zero.	9.7
9.6.2.5#11	The Lane Speed Exponent (bits 5:4) must be identical for both bmSublinkSpeedAttr fields with the same SSID.	9.7
9.6.2.5#12	The Link Protocol (bits 15:14) must be identical for both bmSublinkSpeedAttr fields with the same SSID.	9.7
9.6.2.5#13	The Lane Speed Mantissa (bits 31:16) must be identical for both bmSublinkSpeedAttr fields with the same SSID.	9.7
9.6.2.5#14	If a device implements a SuperSpeedPlus Capability descriptor then the SuperSpeedPlus Capability descriptor returned in response to a GetDescriptor(BOS) request must return value of 16 (DEVICE_CAPABILITY) in the bDescriptorType field.	9.7

Assertion #	Assertion Description	Test #
9.6.2.5#15	If a device implements a SuperSpeedPlus Capability descriptor then the SuperSpeedPlus Capability descriptor returned in response to a GetDescriptor(BOS) request must return value of 0x0A (SUPERSPEED_PLUS) in the DevCapabilityType field	9.7
9.6.2.5#16	The Link Protocol (bits 15:14) must be less than 2	
9.6.2.6#1	The length of a Precision Time Measurement capability descriptor must be 3.	9.7
9.6.2.6#2	If a device implements a Precision Time Measurement descriptor then the Precision Time Measurement descriptor returned in response to a GetDescriptor(BOS) request must return value of 16 (DEVICE_CAPABILITY) in the bDescriptorType field.	9.7
9.6.2.6#3	If a device implements a Precision Time Measurement descriptor then the Precision Time Measurement descriptor returned in response to a GetDescriptor(BOS) request must return value of 0x0B (PRECISION_TIME_MEASUREMENT) in the DevCapabilityType field	9.7
Subsection reference: 9.6.2.7 Configuration Summary Descriptor		
9.6.2.7#1	A Configuration Summary Descriptor shall not be present unless the device has more than one configuration.	
9.6.2.7#2	If implemented, each function presented by the device that does not appear in all configurations shall be represented by a separate Configuration Summary Descriptor. A function that appears in all configurations may optionally be represented by a Configuration Summary Descriptor.	
9.6.2.7#3	Configuration Summary Descriptors should be included in the BOS descriptor in order of descending preference.	
9.6.2.7#4	The bLength field of the Configuration Summary Descriptor shall be set to the size of the descriptor, which is 9 plus the value of the bConfigurationCount field.	
9.6.2.7#5	The bDescriptorType field of the Configuration Summary Descriptor shall be set to 10h (DEVICE_CAPABILITY).	
9.6.2.7#6	The bDevCapabilityType field of the Configuration Summary Descriptor shall be set to 10h (CONFIGURATION SUMMARY).	
9.6.2.7#7	The bcdVersion field of the Configuration Summary Descriptor shall be set to 0100h.	

Assertion #	Assertion Description	Test #
9.6.2.7#8	The bClass field, the bSubClass field, and the bProtocol field of the Configuration Summary Descriptor shall together be equal to the corresponding Class/Subclass/Protocol triple of a USB Function that belongs to one or more configurations.	
9.6.2.7#9	The bConfigurationCount field of the Configuration Summary Descriptor contains the number of bConfigurationIndex fields the Configuration Summary Descriptor contains.	
9.6.2.7#10	Each bConfigurationIndex in the Configuration Summary Descriptor contains the index for a configuration containing this class/subclass/protocol.	
Subsection reference: 9.6.3 Configuration Descriptor		
9.6.3#1	A device must have at least one configuration.	Test Initialization, 9.1
9.6.3#2	The descriptor returned in response to a GetDescriptor(Configuration) request must return the value of 02 (CONFIGURATION) in the bDescriptorType field.	9.2
9.6.3#3	The descriptor returned in response to a GetDescriptor(Configuration) request must have the value of the total length of data returned for this configuration in the Total Length field.	9.2
9.6.3#4	The descriptor returned in response to a GetDescriptor(Configuration) request must return number of interfaces supported by this configuration in the Number of Interfaces field.	9.2
9.6.3#5	Bit D6 in the attributes field of a descriptor returned in response to a GetDescriptor(Configuration) request must be set to 1 if the device is self-powered	Interop Test Procedure
9.6.3#6	Bit D6 in the attributes field of a descriptor returned in response to a GetDescriptor(Configuration) request must be set to 0 if the device is bus-powered	Interop Test Procedure
9.6.3#7	Bit D5 in the attributes field of a descriptor returned in response to a GetDescriptor(Configuration) request must be set to 1 if the device configuration supports remote wakeup	9.2
9.6.3#8	Bit D5 in the attributes field of a descriptor returned in response to a GetDescriptor(Configuration) request must be set to 0 if the device configuration does not support remote wakeup	9.15

Assertion #	Assertion Description	Test #
9.6.3#9	A bus-powered device cannot draw zero power.	9.2, Interop Test Procedure
9.6.3#10	The descriptor returned in response to a GetDescriptor(Configuration)request cannot contain a descriptor of type other speed configuration.	9.2
9.6.3#11	The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request cannot contain a descriptor of type configuration.	Chap 9 Tests for 2.0 devices
9.6.3#12	The configuration descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request must have a length of 0x09.	Chap 9 Tests for 2.0 devices
9.6.3#13	The descriptor returned in response to a GetDescriptor(Configuration) request must contain the number of interfaces descriptors reported in the configuration descriptor.	9.2
9.6.3#14	The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request must contain the number of interfaces descriptors reported in the configuration descriptor.	Chap 9 Tests for 2.0 devices
9.6.3#15	The descriptor returned in response to a GetDescriptor(Configuration) request must contain the number of endpoint descriptors reported in the contained interface descriptors.	9.2
9.6.3#16	The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request must contain the number of endpoint descriptors reported in the contained interface descriptors.	Chap 9 Tests for 2.0 devices
9.6.3#17	The descriptor returned in response to a GetDescriptor(OtherSpeedConfiguration)request must the value of 07 (OTHER_SPEED_CONFIGURATION) in the bDescriptorType field.	Chap 9 Tests for 2.0 devices
9.6.3#18	Bits 0 through 4 must be set to zero in the bmAttributes field of a Configuration descriptor.	9.2
9.6.3#19	Bit 7 must be set to one in the bmAttributes field of a Configuration descriptor.	9.2
9.6.3#20	The number of interfaces cannot be a zero in a [OtherSpeed]Configuration descriptor.	Chap 9 Tests for 2.0 devices
9.6.3#21	A High/Full/Low Speed device operating in self-powered mode cannot draw more than 100ma from the USB bus.	Interop Test Procedure
9.6.3#22	A High/Full/Low Speed device operating in bus-powered mode cannot draw more than 500ma.	Interop Test Procedure

Assertion #	Assertion Description	Test #
9.6.3#23	An Enhanced SuperSpeed device operating in self-powered mode cannot draw more than 150ma from the USB bus.	Interop Test Procedure
9.6.3#24	An Enhanced SuperSpeed device operating in bus-powered mode cannot draw more than 900ma.	Interop Test Procedure
9.6.3#25	A device that reports in its Configuration descriptor that it is self-powered must report in GetStatus() that it is self-powered, and a device that reports in its Configuration descriptor that it is bus-powered must report in GetStatus() that it is bus-powered.	9.2
9.6.3#26	When a device that can operate without its external power source is disconnected from its external power source the device shall disconnect and reconnect as a bus-powered device.	Interop Test Procedure
9.6.3#27	When a device that cannot operate without its external power source is disconnected from its external power source the device shall return to the Powered State	Interop Test Procedure
9.6.3#28	The configuration descriptor returned in response to a GetDescriptor(Configuration) request must have a length of 0x09.	9.2
Subsection reference: 9.6.4 Interface Association Descriptor		
9.6.4#1	The Interface Association descriptor returned in response to a GetDescriptor(Configuration) request must return the value of 11 (INTERFACE_ASSOCIATION) in the bDescriptorType field.	9.3
9.6.4#2	The Interface Association descriptor returned in response to a GetDescriptor(Configuration) request must return interface number of the first interface which is associated with this function in the bFirstInterface Field	9.3
9.6.4#3	The Interface Association descriptor returned in response to a GetDescriptor(Configuration) request must return the number of contiguous interfaces that are associated with this function in the bInterfaceCount field.	9.3, 9.14
9.6.4#4	The Interface Association descriptor returned in response to a GetDescriptor(Configuration) request must not contain zero in the bFunctionClass field.	9.3
9.6.4#5	The Interface Association descriptor returned in response to a GetDescriptor(Configuration) request must contain FFh in the bFunctionClass field if the function class is vendor specific.	9.3

Assertion #	Assertion Description	Test #
9.6.4#6	The Interface Association descriptor returned in response to a GetDescriptor(Configuration) request must return the value of 0x08 in the bLength field	9.3
9.6.4#7	The Interface Association descriptor returned in response to a GetDescriptor(Configuration) request must not contain 0x00 in the bFirstInterface field	9.3
Subsection reference: 9.6.5 Interface Descriptor		
9.6.5#1	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must return the value of 04 (INTERFACE) in the bDescriptorType field	9.4
9.6.5#2	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must return interface number of the interface (which is the zero based value identifying the index in the array of concurrent interfaces supported by current configuration) in the Interface Number field	9.4
9.6.5#3	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must return the value used to select the current alternate setting for the interface identified in the Interface number field, in the Alternate Setting field	9.4
9.6.5#4	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must return the number of endpoints used by the current interface (excluding the Default Control Pipe) in the Number of Endpoints field.	9.9
9.6.5#5	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must contain FFH in the Interface class field if the interface class is vendor specific.	9.4
9.6.5#6	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must contain zero in the Interface Sub Class field if the Interface Class field contains zero.	9.4
9.6.5#7	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must contain zero in the Interface Protocol field if the device does not use a class specific protocol on this interface.	9.4
9.6.5#8	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must contain FFH in the Interface Protocol field if the device uses a vendor specific protocol for this interface.	9.4

Assertion #	Assertion Description	Test #
9.6.5#9	The Interface descriptor returned in response to a GetDescriptor(Configuration) request must contain the index of string descriptor describing this interface in the Interface field.	9.4
9.6.5#10	Alternate settings for a given interface must be in sequential order.	9.4
9.6.5#11	Interface numbers must be in sequential order.	9.4
9.6.5#12	The first interface must have an interface number of 0x0 and an alternate setting of 0x0.	9.4
9.6.5#13	The interface number cannot be greater than or equal to the number of interfaces reported in the configuration descriptor.	9.4,
9.6.5#14	An Interface descriptor must have exactly the number of endpoint descriptors it specifies in the bNumEndpoints field.	9.4, 9.5, 9.6, 9.9
9.6.5#15	An Interface must have at least one setting with an Alternate Setting set to zero.	9.4
9.6.5#16	An interface descriptor must have a length of 0x09.	9.4
9.6.5#17	Each configuration must have at least one interface.	9.4, 9.5, 9.6, 9.9
Subsection reference: 9.6.6 Endpoint Descriptor		
9.6.6#1	Every endpoint used for an interface must have its own descriptor.	9.5, 9.6, 9.9
9.6.6#2	The Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must contain the size of the descriptor in bytes in the length field.	9.5
9.6.6#3	The Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must contain 05 (ENDPOINT) in the bDescriptorType field.	9.5
9.6.6#4	Bits 3:0 in the Endpoint address field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must contain the endpoint number.	Implicit in Interop Test Procedure
9.6.6#5	Bits 6:4 in the Endpoint address field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be set to zero.	9.5
9.6.6#6	Bit 7 in the Endpoint address field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request for a non-control endpoint must be one for an IN endpoint.	Implicit in Interop Test Procedure

Assertion #	Assertion Description	Test #
9.6.6#7	Bits 1:0 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be 00 for a Control endpoint.	Implicit in Interop Test Procedure
9.6.6#8	Bits 1:0 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be 01 for an Isochronous endpoint.	Implicit in Interop Test Procedure
9.6.6#9	Bits 1:0 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be 10 for a Bulk endpoint.	Implicit in Interop Test Procedure
9.6.6#10	Bits 1:0 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be 11 for an Interrupt endpoint.	Implicit in Interop Test Procedure
9.6.6#11	Bits 3:2 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are Reserved and must be set to zero for an Interrupt endpoint.	9.5
9.6.6#12	Bits 5:4 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be 00 for Periodic Usage Type for an Interrupt endpoint.	Implicit in Interop Test Procedure
9.6.6#13	Bits 5:4 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be 01 for Notification Usage Type for an Interrupt endpoint.	Implicit in Interop Test Procedure
9.6.6#14	Bits 3:2 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are 00 if there is no synchronization for an Isochronous transfer.	Implicit in Interop Test Procedure
9.6.6#15	Bits 3:2 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are 01 if the synchronization is Asynchronous for an Isochronous transfer.	Implicit in Interop Test Procedure
9.6.6#16	Bits 3:2 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are 10 if the synchronization is Adaptive for an Isochronous transfer.	Implicit in Interop Test Procedure
9.6.6#17	Bits 3:2 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are 11 if the synchronization is Synchronous for an Isochronous transfer.	Implicit in Interop Test Procedure

Assertion #	Assertion Description	Test #
9.6.6#18	Bits 5:4 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are 00 if the Usage Type is Data Endpoint for an Isochronous transfer.	Implicit in Interop Test Procedure
9.6.6#19	Bits 5:4 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are 01 if the Usage Type is Feedback Endpoint for an Isochronous transfer.	Implicit in Interop Test Procedure
9.6.6#20	Bits 5:4 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are 10 if the Usage Type is Implicit feedback Data Endpoint for an Isochronous transfer.	Implicit in Interop Test Procedure
9.6.6#21	Bits 4 and 5 cannot BOTH be set in the bmAttributes field of an Isochronous Endpoint descriptor.	9.5
9.6.6#22	Bits 5:2 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are Reserved and must be set to zero if the Endpoint is not Isochronous or Interrupt	9.5
9.6.6#23	Bits 7:6 in the Attributes field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request are Reserved and must be set to zero.	9.5
9.6.6#24	Max Packet Size field for SuperSpeed Control Endpoints must be set to 512.	9.5
9.6.6#25	Max Packet Size field for SuperSpeed Bulk Endpoints must be set to 1024.	9.5
9.6.6#26	Max Packet Size field for SuperSpeed Interrupt Endpoints must be set to 1024 if this endpoint defines a value in bMaxBurst field greater than zero	9.5
9.6.6#27	Max Packet Size field for SuperSpeed Isochronous Endpoints must be set to 1024 if this endpoint defines a value in bMaxBurst field in the Endpoint companion descriptor field greater than zero	9.5
9.6.6#28	Max Packet Size field for SuperSpeed Isochronous Endpoints must be between 0 to 1024 if this endpoint defines a value in bMaxBurst field in the Endpoint companion descriptor is set to zero	9.5
9.6.6#29	Max Packet Size field for SuperSpeed Interrupt Endpoints must be between 1 to 1024 if this endpoint defines a value in bMaxBurst field in the Endpoint companion descriptor is set to zero	9.5

Assertion #	Assertion Description	Test #
9.6.6#30	The bInterval field in the endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be between 1 to 16 for SuperSpeed Isochronous and Interrupt Endpoints.	9.5
9.6.6#31	The bInterval field in the endpoint descriptor returned in response to GetDescriptor(Configuration) is Reserved and must be set to zero for SuperSpeed Bulk or Control Endpoints.	9.5
9.6.6#32	Only the default Control endpoint can have an address of 0x00.	Test Initiailization, 9.5
9.6.6#33	The MaxPacketSize of a Control endpoint must be 0x08 for devices operating at Low Speed.	Chap 9 Tests for 2.0 devices
9.6.6#34	Bits 15:13 in the wMaxPacketSize field in the endpoint descriptor returned in response to a GetDescriptor(Configuration) request are set to zero in non-Enhanced SuperSpeed devices.	Chap 9 Tests for 2.0 devices
9.6.6#35	Legal mult [additional transaction opportunities per microframe] values are 0x00/0x01/0x02 for a High Speed Isochronous/Interrupt Endpoint descriptor.	Chap 9 Tests for 2.0 devices
9.6.6#36	Bits 11 and 12 must be set to zero in the wMaxPacketSize field of a Control/Bulk Endpoint descriptor for a non-Enhanced SuperSpeed device	Chap 9 Tests for 2.0 devices
9.6.6#37	Bits 11 and 12 must be set to zero in the wMaxPacketSize field of an Endpoint descriptor for Low Speed and Full Speed devices.	Chap 9 Tests for 2.0 devices
9.6.6#38	USB 1.x compliant devices must have a value of 0x01 in the bInterval field of an Isochronous Endpoint descriptor.	Chap 9 Tests for 2.0 devices
9.6.6#39	A Low Speed Interrupt endpoint must have a value greater than 10 in the bInterval field.	Chap 9 Tests for 2.0 devices
9.6.6#40	Devices operating at Low Speed cannot have Isochronous endpoints.	Chap 9 Tests for 2.0 devices
9.6.6#41	A Full Speed Isochronous endpoint must have a MaxPacketSize between 0 and 1023.	Chap 9 Tests for 2.0 devices
9.6.6#42	A High Speed Interrupt/Isochronous endpoint must have a MaxPacketSize between 0 and 1024 when the Mult [additional transaction opportunities per microframe] value is zero.	Chap 9 Tests for 2.0 devices
9.6.6#43	A High Speed Interrupt/Isochronous endpoint must have a MaxPacketSize between 513 and 1024 and bInterval value of 1 when the Mult [additional transaction opportunities per microframe] value is one.	Chap 9 Tests for 2.0 devices

Assertion #	Assertion Description	Test #
9.6.6#44	A High Speed Interrupt/Isochronous endpoint must have a MaxPacketSize between 683 and 1024 and bInterval value of 1 when the Mult [additional transaction opportunities per microframe] value is two.	Chap 9 Tests for 2.0 devices
9.6.6#45	Devices operating at Low Speed cannot have Bulk endpoints.	Chap 9 Tests for 2.0 devices
9.6.6#46	A Full Speed Bulk endpoint must have a MaxPacketSize of 0x08/0x10/0x20/0x40.	Chap 9 Tests for 2.0 devices
9.6.6#47	A High Speed Bulk endpoint must have a MaxPacketSize of 0x200.	Chap 9 Tests for 2.0 devices
9.6.6#48	A Low Speed Interrupt endpoint must have a MaxPacketSize less than or equal to 0x08.	Chap 9 Tests for 2.0 devices
9.6.6#49	A Full Speed Interrupt endpoint must have a MaxPacketSize less than or equal to 0x40.	Chap 9 Tests for 2.0 devices
9.6.6#50	The bInterval field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be between 1 to 16 for SuperSpeed Isochronous endpoints if the endpoint defines a value greater than 0 in the bAttributes field bits 1:0 (Mult) in the Endpoint Companion descriptor.	9.5
9.6.6#51	Bits 5:4 in the Attributes field in the endpoint descriptor returned in response to a GetDescriptor(Configuration) request cannot be set to 11 for an Interrupt endpoint.	9.5
9.6.6#52	Bits 5:4 in the Attributes field.in the endpoint descriptor returned in response to a GetDescriptor(Configuration) request cannot be set to 10 for an Interrupt endpoint.	9.5
9.6.6#53	Feedback endpoint must always send data in the opposite direction from the data endpoint it services	
9.6.6#54	An explicit feedback endpoint shall have Bits 3:2 of the bmAttributes field of the endpoint descriptor set to 00	9.5
9.6.6#55	If multiple data endpoints are to be serviced by the same feedback endpoint, the data endpoints shall have ascending ordered, but not necessarily consecutive, endpoint numbers	
9.6.6#56	Bit 7 in the Endpoint address field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request for a non-control endpoint must be zero for an OUT endpoint	Implicit in Interop Test Procedure

Assertion #	Assertion Description	Test #
9.6.6#57	The bInterval field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be between 1 to 16 for SuperSpeed Interrupt endpoints if the endpoint defines a value greater than 0 in the bAttributes field bits 1:0 (Mult) in the Endpoint Companion descriptor.	9.5
9.6.6#58	The bInterval field in the Endpoint descriptor returned in response to a GetDescriptor(Configuration) request must be between 8 to 16 for SuperSpeed Notification type Interrupt endpoints if the endpoint defines a value greater than 0 in the bAttributes field bits 1:0 (Mult) in the Endpoint Companion descriptor.	9.5
Subsection reference: 9.6.7 SuperSpeed Endpoint Companion Descriptor		
9.6.7#1	All Enhanced SuperSpeed Devices must have Endpoint Companion descriptors for each of the endpoints.	9.5, 9.6
9.6.7#2	Enhanced SuperSpeed devices shall return Endpoint Companion descriptors for each of the endpoints in that interface to return additional information about its endpoint capabilities in response to the GetDescriptor(Configuration) request.	9.6
9.6.7#3	The SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request must contain the size of the descriptor set to 0x6 in the length field.	9.6
9.6.7#4	The SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request must contain 48 (SUPERSPEED_USB_ENDPOINT_COMPANION) in the descriptor Type field.	9.6
9.6.7#5	Each SuperSpeed endpoint described in an interface must be immediately followed by a SuperSpeed Endpoint Companion descriptor except for the default Control Pipe	9.5, 9.6
9.6.7#6	The value of MaxBurst must be in the range of 0 to 15 which indicates the number of packets the endpoint can send or receive as a part of a burst for Bulk, Interrupt and Isochronous Endpoints.	9.6
9.6.7#7	The value of MaxBurst must be zero for Control Endpoints.	9.6
9.6.7#8	Bits 4:0 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request must be zero if the endpoint does not define streams for Bulk Endpoint.	Implicit in Interop Test Procedure

Assertion #	**Assertion Description**	**Test #**
9.6.7#9	Bits 7:5 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request are Reserved and must be set to zero if it is a Bulk Endpoint.	9.6
9.6.7#10	Bits 7:0 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request are Reserved and must be set to zero if it is a Control or Interrupt Endpoint.	9.6
9.6.7#11	Bits 6:2 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request are Reserved and must be set to zero if it is an Isochronous Endpoint.	9.6
9.6.7#12	Bits 1:0 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request are set to the Mult value supported by the device. A device must send (Mult+1)*(bMaxBurst+1) packets or less per service interval if it is an Isochronous Endpoint.	Implicit in Interop Test Procedure
9.6.7#13	Bits 1:0 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request must have a maximum value of 2 for an Isochronous Endpoint.	9.6
9.6.7#14	Only SuperSpeed devices can have SuperSpeed Endpoint Companion Descriptors.	Chap 9 Tests for 2.0 devices
9.6.7#15	wBytesPerInterval must be zero for Control and Bulk Endpoints.	9.6
9.6.7#16	The value represented by bmAttributes Bits 4:0 by the SuperSpeed Endpoint companion descriptor for Bulk Endpoints shall properly reflect the number of streams supported by the device.	Partially Tested in Interop Test Procedure
9.6.7#17	Bits 1:0 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request are set to 01 if the device supports two Bursts of bMaxBurst packets per service interval if it is an Isochronous Endpoint.	Implicit in Interop Test Procedure
9.6.7#18	Bits 1:0 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request are set to 10 if the device supports three Bursts of bMaxBurst packets per service interval if it is an Isochronous Endpoint.	Implicit in Interop Test Procedure

Assertion #	Assertion Description	Test #
9.6.7#19	Bits 4:0 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request must be between zero and 16d for Bulk Endpoint.	9.6
9.6.7#20	For isochronous endpoints, if bit 7 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request is set, then a SuperSpeedPlus Isochronous Endpoint Companion descriptor must immediately follow this descriptor.	9.28
9.6.7#21	For isochronous endpoints, if bit 7 of the Attributes field in the SuperSpeed Endpoint Companion descriptor is set, then wBytesPerInterval must be set to 1.	9.6
9.6.7#22	For non-isochronous endpoints, a SuperSpeedPlus Isoch Endpoint Companion descriptor must not follow a SuperSpeed Endpoint Companion descriptor.	9.28
Subsection reference: 9.6.8 SuperSpeedPlus Isochronous Endpoint Companion Descriptor		
9.6.8#1	A SuperSpeedPlus Isochronous Endpoint Companion descriptor shall only be returned by a device that is operating at above gen 1 speed.	9.28
9.6.8#2	The length field of a SuperSpeedPlus Isochronous Endpoint Companion descriptor must be 8.	9.28
9.6.8#3	The reserved field of a SuperSpeedPlus Isochronous Endpoint Companion descriptor must be zero.	9.28
9.6.8#4	The dwBytesPerInterval field of a SuperSpeedPlus Isochronous Endpoint Companion descriptor must be greater than 48K (C000h).	9.28
9.6.8#5	The dwBytesPerInterval field of a SuperSpeedPlus Isochronous Endpoint Companion descriptor shall be less than or equal to (MAX_ISO_BYTES_PER_B1_GEN1 x Number_of_Lanes x Speed_Mantissa) / LANE_SPEED_MANTISSA_GEN1)	9.28
Subsection reference: 9.6.9 String Descriptor		
9.6.9#1	String descriptors shall all be UNICODE UTF16LE encoded	
9.6.9#2	A device that omits all string descriptors shall not return an array of LANGID codes.	
9.6.9#3	The size of the string descriptor in bytes is computed by subtracting 2 from the value of the first byte of the descriptor.	

Assertion #	Assertion Description	Test #
9.6.9#4	bLength of a UNICODE string descriptor is 2 + length in bytes of the string.	
9.6.9#5	String descriptors are not NULL terminated.	
9.6.9#6	The bDescriptorType of all string descriptors must be 3.	9.8
9.6.9#7	String descriptor Zero for all languages returns a string descriptor that contains an array of 2-byte LANGID codes supported by the device.	9.8
Subsection reference: 10 Hub, Host Downstream Port, and Device Upstream Port Specification		
10#1	A bus-powered hub is one that uses Standard USB power.	
10#2	A self-powered hub draws power from an external source via non-USB connector or USB PD (upstream or downstream port) or USB Type-C current (upstream)	

3. Test Descriptions for Chapter 9

General Test Initialization

All Chapter 9 tests follow the same initialization procedure. At the beginning of a test run, the host controller is reset and devices attached to the host are enumerated. If attached devices fail initialization assertions, the test tool will not be able to run the test. Note that the host controller is not reset in between tests if more than 1 test is selected.

Assertions Used in Test Initialization

9.1.1#1, 9.1.1#2, 9.1.1.3#1, 9.1.1.4#1, 9.1.1.5#1, 9.2.6.1#1, 9.2.6.3#1 9.2.6.4#1, 9.3#1, 9.4.3#5, 9.4.3#6 9.4.3#10, 9.4.3#10, 9.4.3#11

TD 9.1 Device Descriptor Test

Assertions Used in Test

9.4.3#2, 9.6.1#1, 9.6.1#2, 9.6.1#6, 9.6.1#7, 9.6.1#8, 9.6.1#9, 9.6.1#10, 9.6.3#1

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to One (Device).
    - wIndex – set to Zero.
    - wLength – 18d (All of the Device Descriptor).

   Test fails if the device returns bLength set to anything but 18d (12H).
4. Parse the data returned.
5. Check the returned descriptor for the following values:
    - bDescriptorType must be 01H (DEVICE)

- bcdUSB.hibyte must be 03H
- bcdUSB.lowbyte must be 20H
- bMaxPacketSize0 must be 09H
- idVendor is in the list of valid entries maintained by the USB-IF.
- bDeviceClass is 0 or FFH or is on a list of assigned class codes maintained by the USB-IF.
- If bDeviceClass is 0 then bDeviceSubClass must also be 0
- Run TD 9.8 String Descriptor Test on the returned iManufacturer
- Run TD 9.8 String Descriptor Test on the returned iProduct
- Run TD 9.8 String Descriptor Test on the returned iSerialNumber

Test fails if any of the values are not as specified.

6. If the device supports multiple configurations, repeat this test for each configuration
7. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.2 Standard Configuration Descriptor Test

This test verifies the fields that come from the device are formatted in compliance with the specification and have appropriate values.

Assertions Used in Test

9.4.3#6, 9.4.3#10, 9.4.3#11, 9.4.5#25, 9.6.3#2, 9.6.3#4, 9.6.3#7, 9.6.3#9, 9.6.3#10, 9.6.3#13, 9.6.3#15, 9.6.3#18, 9.6.3#19, 9.6.3#23, 9.6.3#24, 9.6.3#25, 9.6.3#28

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d (All of the Configuration Descriptor).

Test fails if the device returns bLength set to anything but 9d.

3. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the Configuration Descriptor Set).
4. Parse the data returned, only keeping the Standard Configuration Descriptor.
5. Check the returned descriptor for the following values:
    - Configuration.bDescriptorType must be two (2d).
    - Configuration.bmAttributes bit 7 must be one (1b).
    - Configuration.bmAttributes bit 6 must be one (1b) if device is self powered
    - Configuration.bmAttributes bit 5 must be one (1b) if device configuration supports remote wakeup.
    - Configuration.bmAttributes bits 4-0 must be zero (00000b).
    - Configuration.bMaxPower must be <=112d if device is bus powered (112d * 8mA = 896mA)
    - Configuration.bMaxPower must be <= 18d if device is self powered (18d * 8mA = 144mA)
    - Run TD 9.8 String Descriptor Test on the returned Configuration.iConfiguration

Test fails if any of the values are not as specified.

6. Test fails if device has a Device Qualifier Descriptor.
7. Test fails if device has a Other Speed Configuration Descriptor

8. If the device supports multiple configurations, repeat this test for each configuration
9. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.3 Standard Interface Association Descriptor Test

This test verifies that the device under test reports interface association descriptors in compliance with the specification.

Assertions Used in Test

9.6.4#1, 9.6.4#3, 9.6.4#4, 9.6.4#6

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d (All of the Configuration Descriptor).

Test fails if the device returns bLength set to anything but 9d.

3. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the Configuration Descriptor Set).
4. Parse the data returned, only keeping the Standard Interface Association Descriptor.
5. Check the returned descriptor for the following values:
    - InterfaceAssociation.bDescriptorType must be eleven (11d).
    - InterfaceAssociation.bInterfaceCount must be the number of contiguous interfaces associated with this function
    - InterfaceAssociation.bFunctionClass must not be zero
    - Run TD 9.8 String Descriptor Test on the returned InterfaceAssociation.iFunction

Test fails if any of the values are not as specified.

6. If the device supports multiple configurations, repeat this test for each configuration
7. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.4 Standard Interface Descriptor Test

This test verifies that the device under test reports interface descriptors in compliance with the specification.

Assertions Used in Test

4.4.7.1#1, 4.4.8.1#1, 9.4.4#1, 9.4.4#2, 9.4.4#5, 9.4.10#1, 9.6.5#1, 9.6.5#2, 9.6.5#3, 9.6.5#5, 9.6.5#6, 9.6.5#7, 9.6.5#8, 9.6.5#9, 9.6.5#10, 9.6.5#11, 9.6.5#12, 9.6.5#13, 9.6.5#14, 9.6.5#15, 9.6.5#16, 9.6.5#17

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.

2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d

Test fails if the device returns bLength set to anything but 9d.

3. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the Configuration Descriptor Set).
4. Perform the following bandwidth check for all interface and endpoint descriptors returned by the device:
    - Every default interface (alternate setting zero) must not contain an Isochronous endpoint with a maximum packet size greater than 0
    - Every default interface (alternate setting zero) must not contain an Interrupt endpoint with a maximum packet size greater than 64d
5. Check that there is at least one interface descriptor in the data returned.
6. Parse the data returned.
7. Check that the first interface descriptor returned has bInterfaceNumber and bAlternateSetting values of zero.
8. Check each of the returned interface descriptor for the following values:
    - Interface.bDescriptorType must be four (4d).
    - If current and previous interface numbers are the same then current alternate setting must be one greater
    - If current and previous interface numbers are not the same then
        - Current interface number must be 1 greater than previous
        - Current alternate setting must be zero
        - Current interface number must be < total number interfaces
    - Interface.bLength must be 9d
    - If Interface Class is vendor specific then Interface.bInterfaceClass must be FFH
    - If Interface.bInterfaceClass is zero then Interface.bInterfaceSubClass must be zero
    - If the device does not use a class specific protocol on this interface then Interface.bInterfaceProtocol field must be zero
    - Run TD 9.8 String Descriptor Test on the returned Interface.iInterface

Test fails if any of the values are not as specified.

9. If the device state is Configured State:
    - Send a GetInterface() and verify that the device is initially configured to alternate setting 0.
        - If the current interface has no alternate settings then the device may STALL this request
    - For every Interface:
        - Send a SetInterface()
            - If the current interface has no alternate settings then the device may STALL this request
        - Send a GetInterface() and verify that the settings are the same
            - If the current interface has no alternate settings then the device may STALL this request
10. If the device supports multiple configurations, repeat this test for each configuration.
11. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.5 Endpoint Descriptor Test

This test verifies that the device under test reports endpoint descriptors in compliance with the specification.

Assertions Used in Test

9.4.3#4, 9.6.5#14, 9.6.5#17,9.6.6#1, 9.6.6#2, 9.6.6#3, 9.6.6#5, 9.6.6#11, 9.6.6#21, 9.6.6#22, 9.6.6#23, 9.6.6#24, 9.6.6#25, 9.6.6#26, 9.6.6#27, 9.6.6#28, 9.6.6#29, 9.6.6#30, 9.6.6#31, 9.6.6#32, 9.6.6#51, 9.6.6#52, 9.6.6#54, 9.6.7#1, 9.6.7#5

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d

Test fails if the device returns bLength set to anything but 9d.

3. Send GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the Configuration Descriptor Set).
4. Issue a valid get device descriptor command to store the bcdUSB value.
5. Parse the data returned, keeping all endpoint descriptors obtained
6. Check the returned descriptor for the following values:

All –

- EndpointDescriptor.bLength must be equal to seven (7d)
- Endpoint.bDescriptorType must be five for ENDPOINT (5d)
- Endpoint.bEndPointAddress must not be 00H or 80H
- Endpoint.bmAttributes bits 7 and 6 must be zero
- Each Endpoint must be directly followed by a SuperSpeed endpoint companion descriptor

If Interrupt –

- Endpoint.bmAttributes, if set to Interrupt, bits 1:0 must be 11b
- Endpoint.bmAttributes, bits 3:2 must be zero (Reserved)
- Endpoint.bmAttributes, bits 5:4 must not be set to 11b
- Endpoint.bmAttributes, bits 5:4 must not be set to 10b
- Each Endpoint descriptor must be followed by a SuperSpeed endpoint companion descriptor
    - if EndpointCompanion.bMaxBurst > 0, then Endpoint.wMaxPacketSize must be 1024d
    - if EndpointCompanion.bMaxBurst = 0, then Endpoint.wMaxPacketSize must be between 1 and 1024d.
- Endpoint.bInterval must be between 1 and 16d

If Control –

- Endpoint.bmAttributes, if set to Control, bits 1:0 must be 00b
- Endpoint.bmAttributes, bits 5:2 must be zero (Reserved)
- Endpoint.wMaxPacketSize must be 512d

, Endpoint.bInterval is Reserved and must be zero.

If Isochronous –

- Endpoint.bmAttributes, if set to Isochronous, bits 1:0 must be 01b
- Endpoint.bmAttributes, bits 5:4 report Usage type
    - If bits 5:4 are 01b for Explicit Feedback endpoint, then bits 3:2 must be 00b
    - Bits 5:4 must never be 11b
- Each Endpoint must be followed by a SuperSpeed endpoint companion descriptor

- if EndpointCompanion.bMaxBurst > 0, then Endpoint.wMaxPacketSize must be 1024d
- if EndpointCompanion.bMaxBurst = 0, then Endpoint.wMaxPacketSize must be between 0 and 1024d.

- Endpoint.bInterval must be between 1 and 16d.
  - if EndpointCompanion.bmAttributes bits 1:0 > 0, then Endpoint.bInterval must be 1d

If Bulk –

- Endpoint.bmAttributes, if set to Bulk, bits 1:0 must be 10b
- Endpoint.bmAttributes, bits 5:2 must be zero (Reserved)
- Endpoint.wMaxPacketSize must be 1024d
- Endpoint.bInterval is Reserved and must be zero.

Test fails if any of the values are not as specified.

7. **Verify that feedback endpoints service endpoints in opposite direction**
8. **If multiple data endpoints are to be serviced by the same feedback endpoint, the data endpoints must have ascending order.**
9. If the device supports multiple configurations, repeat this test for each configuration
10. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.6 SuperSpeed Endpoint Companion Descriptor Test

This test verifies that the device under test reports endpoint companion descriptors in compliance with the specification.

Assertions Used in Test

4.4.7.2#1, 9.4.3#4, 9.6.5#14, 9.6.5#17, 9.6.6#1, 9.6.7#1, 9.6.7#2, 9.6.7#3, 9.6.7#4, 9.6.7#5, 9.6.7#6, 9.6.7#7, 9.6.7#9, 9.6.7#10, 9.6.7#11, 9.6.7#13, 9.6.7#15, 9.6.7#19, 9.6.7#21

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
   - wValue – set to Two (Configuration).
   - wIndex – set to Zero.
   - wLength – wTotalLength (All of the Configuration Descriptor Set).
3. Parse the data returned, keeping all endpoint descriptors obtained
4. Check if each SuperSpeed endpoint described in an interface has a SuperSpeed Endpoint Companion Descriptor except for a Control endpoint
5. Check each of the returned Endpoint descriptors for an Endpoint Companion Descriptor
   - EndpointCompanion.bLength must be equal to six (6d)
   - EndpointCompanion.bDescriptorType must be 48d for SUPERSPEED_USB_ENDPOINT_COMPANION (48d)
   - EndpointCompanion.bMaxBurst must be <= 15d.

If Bulk –

- EndpointCompanion.bmAttributes, bits 4:0 must be <= 16d
- EndpointCompanion.bmAttributes, bits 7:5 are reserved and must be 0
- EndpointCompanion.wBytesPerInterval must be 0

If Control –

- EndpointCompanion.bmAttributes, bits 7:0 are reserved and must be 0
- EndpointCompanion.bMaxBurst must be 0
- EndpointCompanion.wBytesPerInterval must be 0

If Interrupt –
- EndpointCompanion.bmAttributes, bits 7:0 are reserved and must be 0
- EndpointCompanion.bMaxBurst must be <= 2d

If Isochronous –
- EndpointCompanion.bmAttributes, bits 1:0 must be <= 2d
- EndpointCompanion.bmAttributes, bits 6:2 are reserved and must be 0
- If EndpointCompanion.bmAttributes bit 7 is 1, then EndpointCompanion.wBytesPerInterval must be 1.

Test fails if any of the values are not as specified.

6. If the device supports multiple configurations, repeat this test for each configuration
7. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.7 BOS and Device Capability Descriptor Test

This test verifies the fields within the BOS and Device Capability Descriptor from the device are formatted in compliance with the specification and have appropriate values.

Assertions Used in Test

8.5.6.7#1, 9.1.1.3#2, 9.1.1.3#3, 9.4.3#9, 9.6.2#1, 9.6.2#3, 9.6.2#4, 9.6.2#5, 9.6.2#7, 9.6.2.1#1, 9.6.2.1#2, 9.6.2.1#3, 9.6.2.1#4, 9.6.2.1#5, 9.6.2.2#1, 9.6.2.2#2, 9.6.2.2#3, 9.6.2.2#5, 9.6.2.2#9, 9.6.2.2#10, 9.6.2.2#11, 9.6.2.2#12, 9.6.2.2#13, 9.6.2.2#14, 9.6.2.2#15, 9.6.2.2#17, 9.6.2.3#1, 9.6.2.3#3, 9.6.2.3#4, 9.6.2.3#5, 9.6.2.3#7, 9.6.2.4#1, 9.6.2.4#2, 9.6.2.4#3, 9.6.2.4#4, 9.6.2.5#1, 9.6.2.5#2, 9.6.2.5#3, 9.6.2.5#4, 9.6.2.5#5, 9.6.2.5#6, 9.6.2.5#7, 9.6.2.5#8, 9.6.2.5#9, 9.6.2.5#10, 9.6.2.5#11, 9.6.2.5#12, 9.6.2.5#13, 9.6.2.5#14, 9.6.2.5#15 9.6.2.6#16, 9.6.2.6#1, 9.6.2.6#2, 9.6.2.6#3

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to 15d (BOS).
    - wIndex – set to Zero.
    - wLength – 5d.

Test fails if the device returns bLength set to anything but 5d.

3. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to 15d (BOS).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the BOS Descriptor Set).
4. Check the size of the returned descriptor table.

Test fails is the length of the returned descriptor table is different from wTotalLength.

5. Parse the data returned, only keeping the BOS and Device Capability Descriptors.
6. Check the returned BOS descriptor for the following values:
    - bLength reflects the size returned.

- bDescriptorType is 15d (BOS type).
- bNumDeviceCaps reflects the number of separate Device Capability Descriptors found in the BOS, which must be at least 2 for Enhanced SuperSpeed devices, at least to 5 for Enhanced SuperSpeed hubs running at Gen X speeds.

Test fails if any of the values are not as specified.

7. Check the returned device capability descriptors for the following values:.
   - All Enhanced SuperSpeed devices and USB3.1 Hubs must have a USB 2.0 EXTENSION descriptor.
   - All Enhanced SuperSpeed devices and USB3.1 Hubs must have a SUPERSPEED_USB descriptor.
   - All USB3.1 Hubs must have a CONTAINER_ID descriptor.

Test fails if any of the values are not as specified.

8. For USB 2.0 EXTENSION, check the returned Descriptor for the following values:
   - bLength == Size of descriptor (07H)
   - bDescriptorType is 16d (DEVICE_CAPIBILITY)
   - bDevCapabillityType is 02H (USB 2.0 EXTENSION)
   - bmAttributes: Bit 0 is 0
     Bit 1 is 1
     Bits 31:2 are 0

Test fails if any of the values are not as specified.

9. For SUPERSPEED_USB, check the returned Descriptor for the following values:
   - bLength == Size of descriptor (0AH)
   - bDescriptorType is 16d (DEVICE_CAPIBILITY)
   - bDevCapabillityType is 03H (SUPERSPEED_USB)
   - bmAttributes: Bit 0 is 0
     Bit 1 is 1 if it is a LTM capable device
     Bits 7:2 are 0
10. For CONTAINER_ID, check the returned Descriptor for the following values:
    - bLength == Size of descriptor (14H)
    - bDescriptorType is 16d (DEVICE_CAPIBILITY)
    - bDevCapabillityType is 04H (CONTAINER_ID)
    - bReserved is 0

Test fails if any of the values are not as specified.

11. For PLATFORM, check the returned Descriptor for the following values:
    - bLength == Size of descriptor (greater than or equal to 14H)
    - bDescriptorType is 16d (DEVICE_CAPABILITY)
    - bDevCapabillityType is 05H (PLATFORM)
    - bReserved is 0

Test fails if any of the values are not as specified.

12. For SUPERSPEED_PLUS, check the returned Descriptor for the following values:
    - bLength == Size of descriptor (CH + (4H * (bmAttributes.SSAC + 1)))
    - bDescriptorType is 16d (DEVICE_CAPIBILITY)
    - bDevCapabillityType is 0AH (SUPERSPEED_PLUS)
    - The reserved fields (bReserved, wReserved) are 0
    - bmAttributes bits 31:9 are 0.
    - wFunctionalitySupport bits 3:0 is equal to bmSublinkSpeedAttr.SSID, for one of the bmSublinkSpeedAttr fields of this descriptor.
    - wFunctionalitySupport bits 7:4 is 0.
    - wFunctionalitySupport.bits 11:8 is one.
    - wFunctionalitySupport.bits 15:12 is one.
    - For each bmSublinkSpeedAttr:
      - bit 6 is 0.
      - bits 13:8 is 0.

  - For each bmSublinkSpeedAttr.SSID (bits 3:0), there are two bmSublinkSpeedAttr's, the first one has bit 7 == 0, the second one (which immediately follows the first) has bit 7 == 1.
  - For each bmSublinkSpeedAttr pair with the same SSID, bits 5:4 must be identical.
  - For each bmSublinkSpeedAttr pair with the same SSID, bits 15:14 must be identical.
  - For each bmSublinkSpeedAttr pair with the same SSID, bits 31:16 must be identical.

Test fails if any of the values are not as specified.

13. For PRECISION_TIME_MEASUREMENT, check the returned Descriptor for the following values:
    - bLength == Size of descriptor (3H)
    - bDescriptorType is 16d (DEVICE_CAPABILITY)
    - bDevCapabillityType is 05H (PRECISION_TIME_MEASUREMENT)

Test fails if any of the values are not as specified.

14. If the device supports multiple configurations, repeat this test for each configuration
15. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.8 String Descriptor Test

This test is run in the process of running each of the previous descriptor tests. In each descriptor that is not a Device Qualifier or Other Speed descriptor this test is run

Assertions Used in Test

9.4.3#5, 9.6.8#6, 9.6.8#7

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. For each non-zero string descriptor in any of the device's descriptors perform the following steps.
3. Issue a valid Get String Descriptor request with index 0 and a requested length of 500.
4. Perform each of the following checks on the returned language ID table:
    - String.bLength must be at least four (4d)
    - String.bLength must be a multiple of two (2d)
    - String.bDescriptorType must be 3d (STRING)
    - Number of bytes received must match the bLength field of the descriptor
5. For each LangID value in the table issue a valid Get String Descriptor request with the index to test and a requested length of 500.
6. For each of the string descriptors received in step 5 check the following:
    - String.bLength must be at least two (2d)
    - String.bDescriptorType must be 3d (STRING)
    - Number of bytes received must match the bLength field of the descriptor
7. Repeat test with device in the default, address, and configured states.
8. If the test supports multiple configurations repeat test for the configured state for each possible configuration.
9. Print out string descriptors, which must be in UNICODE_UTF16LE
10. Repeat test with the device starting in the following states
    - Default
    - Address
    - Configured

TD.9.9 Halt Endpoint Test

This test checks that all Bulk and Interrupt endpoints can be programmatically halted.

Assertions Used in Test

9.4.3#4, 9.4.4#2, 9.4.5#17, 9.4.5#18, 9.4.5#35, 9.6.5#14, 9.6.5#17, 9.6.6#1

Device States for Test

This test is run with the device in Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d

Test fails if the device returns bLength set to anything but 9d.

3. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the Configuration Descriptor Set).
4. For every interface obtained perform the following:
    - Send a SetInterface()
        - If the current interface has no alternate settings then the device may STALL this request
    - Send a GetInterface() and verify that the settings are the same
        - If the current interface has no alternate settings then the device may STALL this request

.

    - For each endpoint obtained for the above interface perform the following checks if the endpoint is Bulk or Interrupt:
        - Issue a valid GetStatus() request for the endpoint.
        - If the endpoint is halted issue a valid Clear Feature request with feature selector ENDPOINT_HALT. Then issue a valid GetStatus() request for the endpoint and verify that the endpoint no longer reports halt.
        - Issue a valid SetFeature() request with feature selector ENDPOINT_HALT
        - Issue a valid GetStatus() request for the endpoint.
        - Verify that ENDPOINT_HALT is reported in the status.
        - Issue a valid Clear Feature request with feature selector ENDPOINT_HALT
        - Issue a valid GetStatus() request for the endpoint.
        - Verify that ENDPOINT_HALT is not reported in the status.

    Test fails if any of the above checks fail.

5. If the device supports multiple configurations, repeat this test for each configuration.

TD.9.10 Bad Descriptor Test

This test checks that a device properly handles a Get Descriptor() request when the Descriptor type is invalid.

Assertions Used in Test

9.4#1, 9.4#2, 9.4.3#1, 9.4.3#2, 9.4.5#25

Device States for Test

This test is run with the device in Address and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Issue a GetDescriptor() request with a with a descriptor type parameter that is invalid. Invalid, means that it is not within the set of defined values in the USB 3.1 specification or any class specifications.
2. Verify that the devic99e responded with a STALL.

3. Issue a GetStatus() request to Endpoint 0, and make sure the device has not halted endpoint 0. If this is successful, skip to step 5.
4. Re-enumerate the device. If this fails then the test aborts.
5. Issue a GetDescriptor(Device) request to the device, to make sure the device is still responding. If this is successful, skip step 7.
6. Re-enumerate the device. If this fails then the test aborts.
7. If the device supports multiple configurations, repeat this test for each configuration

TD.9.11 Bad Feature Test

This test checks that a device properly handles a SetFeature() request when the feature selector is invalid.

Assertions Used in Test

9.4#1, 9.4#2, 9.4.1#1, 9.4.3#2, 9.4.5#25, 9.4.9#2

Device States for Test

This test is run with the device in Address and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Issue a SetFeature() request with a with a feature selector parameter that is invalid. Invalid, means that it is not within the set of defined values in the USB 3.1 specification or any class specifications.
2. Verify that the device responded with a STALL.
3. Issue a GetStatus() request to Endpoint 0, and make sure the device has not halted endpoint 0. If this is successful, skip to step 5.
4. Re-enumerate the device. If this fails then the test aborts.
5. Issue a GetDescriptor(Device) request to the device, to make sure the device is still responding. If this is successful, skip to step 7.
6. Re-enumerate the device. If this fails then the test aborts.
7. Issue a Clear Feature request with a with a feature selector parameter that is invalid. Invalid, means that it is not within the set of defined values in the USB 3.1 specification or any class specifications.
8. Verify that the device responded with a STALL.
9. Issue a GetStatus() request to Endpoint 0, and make sure the device has not halted endpoint 0. If this is successful, skip to step 11.
10. Re-enumerate the device. If this fails then the test aborts.
11. Issue a GetDescriptor(Device) request to the device, to make sure the device is still responding. If this is successful, skip step 13.
12. Re-enumerate the device. If this fails then the test aborts.
13. If the device supports multiple configurations, repeat this test for each configuration

TD.9.12 Remote Wakeup Test

This test is not used for 3.1 devices operating at SuperSpeed. See Remote Wakeup test description in USB 2.0 Test Specification.

TD.9.13 Set Configuration Test

This test checks that each device configuration can be set using SetConfiguration() and that the device properly reports its current configuration in response to the GetConfiguration() command.

Assertions Used in Test

9.4.2#1, 9.4.2#2, 9.4.7#1, 9.4.7#3, 9.4.7#4

Device States for Test

This test is run with the device in Address state.

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid SetConfiguration() request using configuration value 0.
2. Issue a valid GetConfiguration() request and verify that the device reports 0.
3. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
   - wValue – high byte set to Two (CONFIGURATION)
     - low byte set to Zero (Configuration Index)
   - wIndex – set to Zero.
   - wLength – 9d
4. Issue a valid SetConfiguration() request using the bConfigurationValue obtained in the previous step.
   Device is now in Configured state
5. Issue a valid GetConfiguration() request and verify that the configuration set in the previous step is reported.
6. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
   - wValue – high byte set to Two (CONFIGURATION)
     - low byte set to the index of configuration to be tested.
   - wIndex – set to Zero.
   - wLength – 9d
7. Issue a valid SetConfiguration() request using the bConfigurationValue obtained in the previous step.
8. Issue a valid GetConfiguration() request and verify that the configuration set in the previous step is reported.
9. Issue a valid SetConfiguration() request using configuration value 0.
   Device is now in Address state.
10. Issue a valid GetConfiguration() request and verify that the device reports 0.
11. Issue a SetConfiguration() request with an invalid configuration Index.
    This invalid SetConfiguration() request should fail.
12. Issue a valid SetConfiguration() request using the bConfigurationValue obtained in step 5.
    Device is now in Configured state.
13. Issue a valid GetConfiguration() request and verify that the configuration set in the previous step is reported.
14. Issue a SetConfiguration() request with an invalid configuration Index.
    This invalid SetConfiguration() request should fail.
15. Issue a valid SetConfiguration() request using configuration value 0.
    Device is now in Address state.
16. Issue a valid GetConfiguration() request and verify that the device reports 0.
17. If the device supports multiple configurations, repeat this test for each configuration

TD.9.14 Suspend/Resume Test

This test checks that a device can be suspended and then resumed and return to normal operation.

Assertions Used in Test

9.4.9#2, 9.1.1#2, 9.4.1#1, 9.1.1.6#2, 9.2.5.2#3, 9.2.5.2#4, 9.4.5#14, 9.4.5#15, 9.4.9#8, 9.4.9#14, 9.4.9#15, 9.4.9#16, 9.4.9#17, 9.4.9#18, 9.4.1#6, 9.4.5#6, 9.4.9#12

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Suspend the parent port (that the device is attached to) to U3.
2 .Allow 10ms for device to settle.
3. Get Port Status bit state.
4. Port should report that it is suspended to U3.
5. Sleep 200ms

6. Send a GetDescriptor(Device) request to the device. This request should fail because the device should be suspended.
7. Resume the port
8. Send a GetDescriptor(Device) request to the device. If this fails for any reason all devices are re-enumerated to restore the initial device state.

Function Suspend (Configured state only)

35. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d

Test fails if the device returns bLength set to anything but 9d.

36. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the Configuration Descriptor Set).
37. For each function obtained perform the following on the first interface of that function:
    - Send a Set Feature(FUNCTION_SUSPEND) and set the function into Low power suspend state
    - Send a Clear Feature(FUNCTION_SUSPEND) to set the function into Normal operation state

Link Inactivity Timeout (Configured state only) – Steps 38-44 are informational only

38. For each function obtained perform the following on the first interface of that function:
    - Send a Set Feature(FUNCTION_SUSPEND) and set the function into Low power suspend state
39. Enable the parent port (that the device is attached to) to accept transitions to U2.
40. Sleep 10ms
41. Get Port Status bit state.
42. Port should report that it is suspended in U2.
43. Set the parent port to reject all transitions to U1 and to U2.
44. For each function obtained perform the following on the first interface of that function:
    - Send a Clear Feature(FUNCTION_SUSPEND) to set the function into Normal operation state

TD.9.15 Function Remote Wakeup Test

This test checks each device configuration supporting function remote wakeup to ensure that the device properly supports the function remote wake.

Assertions Used in Test

9.2.5.2#1, 9.4.3#6, 9.4.9#11, 9.4.7#1, 9.4.1#2, 9.4.5#11, 9.4.9#13, 9.4.5#27, 9.1.1.6#3, 9.2.5.2#2, 9.2.5.2#5, 9.2.6.2#1, 9.4.1#6, 9.4.5#13, 9.4.5#24, 9.4.5#33, 9.4.9#8, 9.4.9#8

Device States for Test

This test is run with the device in the Configured state. For each function that supports Function Remote Wakeup the test is run with function remote wakeup enabled (must work) and disabled (must not work).

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid GetStatus() command to the device.
    - Verify that D1 is 0 for a USB 3.1 device
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d
3. Check the returned descriptor for the following values:

- Configuration.bLength must be 9 (9d)
- Configuration.bDescriptorType must be two (2d)

4. For each function perform the following Function Remote Wake test on that function:

Function Remote Wake

Note: a device class may have additional requirements for enabling and/or detecting Function Remote Wake. If necessary, any required actions should be performed as appropriate in and around the following steps. See the appropriate device class specifications for details.

5. Issue a valid GetStatus() command to the first interface.
   - If bit D0 is not set, then the device does not support the function remote wake capability, and the test is over for this function.
6. Configure Function Remote Wake:
   - If this is a Function Remote Wake ENABLE test, issue a valid SetFeature() command to the first interface with feature selector FUNCTION_SUSPEND and the Low wIndex set to 0x2 for WAKE_ENABLE.
   - If this is a Function Remote Wake DISABLE test, issue a valid SetFeature() command to the first interface with feature selector FUNCTION_SUSPEND and the Low wIndex set to 0x0 for WAKE_DISABLE.
7. Issue a valid GetStatus() command to the first interface.
   - If bit D1 is not set correctly, reflecting ENABLE or DISABLE, test fails.7. Suspend the parent port of the device under test.
8. Sleep for 1 second.
9. Check the parent port status to verify that it has suspended.
10. Prompt the user to generate a remote wakeup event on the device function under test.
11. Wait for 15 seconds for a DEV_NOTIFICATION event to be generated.
12. Check result of a DEV_NOTIFICATION
    - Notification SubType must be 6
    - Notification NotificationType must be 1
    - If this is a wake disabled test, the test fails if it got a notification matching these settings
    - If this is a wake enabled test, it must have gotten at least one notification matching these settings.
13. Test will not access the device during the 15 second wait, so the device must send a wake notification again after tNotification time period. Test fails if only 1 DEV_NOTIFICATION is received.
14. Read the parent port by sending a valid Get Port Status Command.
    - If this is a wake disabled test, PORT_SUSPENDED must be set.
    - If this is a wake enabled test, PORT_SUSPENDED must be cleared.
15. Do GetStatus() call to verify that Function Suspend and Function Remote Wake settings are identical to original settings.

If this is a Function Remote Enable Test:

16. Reset Device. This should reset the Function Remote Wake.
17. Issue GetStatus() to device. Remote Wake bit should be 0.
18. Repeat steps 7-12.
19. Wait for 15 seconds.
20. If there is a DEV_NOTIFICATION received, the test fails.
21. If the device supports multiple configurations, repeat this test for each configuration

TD.9.16 Enumeration Test

This test checks that a device can be enumerated multiple times.

Assertions Used in Test

9.2.6.2#1

Device States for Test

This test is run with the device starting in the Address state.

Overview of Test Steps

The test software performs the following steps.

1. A message informs the user that the test has started and how many enumeration iterations will be done. The user can abort the test if it will take too long. However, this will be a test failure.
2. Repeat for each test iteration:
   - Issue a port reset
   - Re-enumerate the device tree. If device does not re-enumerate, the test fails.

TD.9.17 Other Speed Configuration Descriptor Test

This test is not used for 3.1 devices operating at SuperSpeed. See Other Speed Configuration Descriptor test description in USB 2.0 Test Specification.

TD.9.18 Device Qualifier Descriptor Test

This test is not used for 3.1 devices operating at SuperSpeed. See Device Qualifier Descriptor test description in USB 2.0 Test Specification.

TD.9.19 Control Transfer Timing Test

This test reports the timing of each phase of a control transfer.

Assertions Used in Test

9.2.6.4#2

Device States for Test

This test is run with the device starting in the Configured state.

Overview of Test Steps

1. Issue a GetDescriptor() request to the device.
2. Test will report how much time the Setup, Data & Status phases take to complete.
3. This test is informational only.

TD.9.20 LTM Test

This test checks the LTM feature

Assertions Used in Test

9.4.5#1, 9.4.9#1, 9.4.9#7, 9.4.5#11, 9.4.1#5

Device States for Test

This test is run with the device starting in the Configured state.

Overview of Test Steps

1. Check LTM Capability in SuperSpeed Device Capability descriptor. If bit 1 of bmAttributes is 0, the test PASSES. End the test.

If the device supports LTM, then continue.

2. Issue GetStatus() to first interface of Device.
   - wValue – 0
   - wIndex – 0
   - wLength – 2
3. Verify that bit D4, LTM Enable, is cleared.
4. Issue SetFeature() to device:
   - wValue – 50 (LTM_ENABLE)
   - wIndex – 0
   - wLength – 0

5. Issue GetStatus() to Device.
6. Verify that bit D4, LTM_Enable, is set.
7. Wait 5 seconds for a DEV_NOTIFICATION with type LATENCY_TOLERANCE_MESSAGE (2h).
8. Verify that a DEV_NOTIFICATION is received in 10 milliseconds
9. Issue ClearFeature() to device:
   - wValue – 50 (LTM_ENABLE)
   - wIndex – 0
   - wLength – 0
10. Wait 5 seconds. Make sure no LTM DEV_NOTIFICATIONS were received.

TD.9.21 LPM L 1 Suspend Resume Test

This test is not used for 3.1 devices operating at SuperSpeed. See LPM L1 Suspend Resume test description in USB 2.0 Test Specification.

TD.9.22 Set Feature Test

This test checks the Set Feature command

Assertions Used in Test

9.4.5#7, 9.4.5#8, 9.4.5#9, 9.4.5#21, 9.4.5#22, 9.4.5#23, 9.4.9#6, 9.4.5#23

Device States for Test

This test is run with the device starting in the Configured state.

Overview of Test Steps

1. Check LTM Capability in SuperSpeed Device Capability descriptor.
   - If bit 1 of bmAttributes is 0, then LTM is NOT SUPPORTED.
   - If bit 1 of bmAttributes is 1, then LTM is SUPPORTED.

2. Check LDM Capability in BOS descriptor:
   - If the Precision Time Management Capability descriptor exists, then LDM is SUPPORTED.
   - If it does not exist then LDM is NOT SUPPORTED and the LDM portions of the test are NOT RUN.

3. If LDM is supported, then check whether the Device Under Test is directly downstream of a gen 1 speed host.
   - If this is true then the LDM portions of the test are NOT RUN.
   - If this is true, then display a warning to the user that in order to run the LDM portions of this test at gen 1 speed that this test must be run with the device directly downstream of a gen 2 hub that is downstream of a gen 1 host port.

4. Issue GetStatus() to Device.
   - wValue – 0
   - wIndex – 0
   - wLength – 2
5. Verify that bit D2, U1 Enable, is cleared.
6. Issue SetFeature() to device:
   - wValue – 48 (U1_ENABLE)
   - wIndex – 0
   - wLength – 0
7. Issue GetStatus() to Device.
8. Verify that bit D2, U1 Enable, is set.
9. Issue ClearFeature() to device:
   - wValue – 48 (U1_ENABLE)

- wIndex – 0
- wLength – 0

10. Issue GetStatus() to Device.
11. Verify that bit D2, U1 Enable, is cleared.
12. Repeat steps 4-11 for U2, with the following differences.
    - In step 5, The U2_Enable status bit is D3.
    - In steps 6 and 9, wValue is 49 (U2_Enable)
13. If LTM is supported, then repeat steps 4-11 with the following differences.
    - In step 5, The LTM_Enable status bit is D4.
    - In steps 6 and 9, wValue is 50 (LTM_Enable)
14. If LDM is to be run, then repeat steps 4-11 with the following differences:
    - In steps 4, 7 & 10 issue GetStatus() with wValue==1, wLength==4.
    - In step 5, The LDM_Enable status bit is D0, which should be set.
    - In steps 6 and 9, wValue is 53 (LDM_Enable)

15. Place device in Addressed state.
16. Issue SetFeature(U1_ENABLE)
17. Verify that SetFeature command failed.
18. Place device in Configured state.
19. Repeat steps 15-18 for U2_ENABLE.
20. If LTM is supported, then repeat steps 15-18 for LTM_ENABLE.
21. If LDM is to be run, then repeat steps 15-18 for LDM_ENABLE, with the following differences:
    - In step 17, verify that SetFeature command succeeds.

22. Issue SetFeature(U1_ENABLE)
23. Issue a GetStatus, and verify that U1_Enable is set.
24. Reset the device (that is, Issue a PortReset command to the parent port of the device).
25. Place device in Configured state.
26. Issue a GetStatus and verify that U1_Enable is cleared.
27. Repeat steps 22-26 for U2_ENABLE.
28. If LTM is supported, then repeat steps 22-26 for LTM_ENABLE.
29. If LDM is to be run, then repeat steps 22-26 for LDM_ENABLE with the following changes.
    - In step 22 issue ClearFeature(LDM_ENABLE).
    - In step 23 verify that entire status (LDM_Enable, LDM_Valid, LDM_Link_Delay, all reserved bits) is zero.
    - In step 26 verify that LDM_Enable is set.

NOTE: For each GetStatus() request, verify that the number of bytes returned is equal to wLength (where wLength==2 if wValue.LoByte=0 (STANDARD_STATUS), and wLength==4 when wValue.LoByte=1 (PTM_STATUS)).

TD.9.23 Reset Device Test

This test checks that device connects at the correct speed depending on the ports enabled

Assertions Used in Test

9.1.1.3#4

Device States for Test

This test is run with the device starting in the Default state.

Overview of Test Steps

1. Connect an Enhanced SuperSpeed device to a SuperSpeed port.
2. Issue a GetDescriptor(BOS) and determine the highest 2.0 supported speed.
3. Disable the SuperSpeed port.

4. Wait for the port connect bit to show a device connected at the highest 2.0 supported speed for this device on the 2.0 port shared by the physical connector.
5. If device does not re-connect at this speed, the test fails.
6. Enable the previously disabled SuperSpeed Port.
7. Issue a 2.0 reset.
8. The device must re-connect on the SuperSpeed Port.

9. Disable the SuperSpeed Port
10. Wait for the port connect bit to show a device connected at the highest 2.0 supported speed for this device on the 2.0 port shared by the physical connector.
11. Issue a 2.0 reset.
12. Wait for the port enable bit to be set on the 2.0 port.
13. Issue a GetDescriptor(device).
14. Enable the previously disabled SuperSpeed Port.
15. Issue a 2.0 reset.
16. The device must re-connect on the SuperSpeed Port.

17. Disable the SuperSpeed Port
18. Wait for the port connect bit to show a device connected at the highest 2.0 supported speed for this device on the 2.0 port shared by the physical connector.
19. Issue a 2.0 reset.
20. Wait for the port enable bit to be set on the 2.0 port.
21. Issue a GetDescriptor(device).
22. Issue a SetAddress() to the device.
23. Enable the previously disabled SuperSpeed Port.
24. Issue a 2.0 reset.
25. The device must re-connect on the SuperSpeed Port.

26. Disable the SuperSpeed Port
27. Wait for the port connect bit to show a device connected at the highest 2.0 supported speed for this device on the 2.0 port shared by the physical connector.
28. Issue a 2.0 reset.
29. Wait for the port enable bit to be set on the 2.0 port.
30. Issue a GetDescriptor(device).
31. Issue a SetAddress() to the device.
32. Issue a SetConfiguration() to the device
33. Enable the previously disabled SuperSpeed Port
34. Issue a 2.0 reset.
35. The device must re-connect on the SuperSpeed Port.

36. Verify that device reports BCD 2.10 when it is operating at HS or FS.

TD.9.24 U1 and U2 Test

This test verifies U1 and U2 behavior. Run at the root port and behind a SS hub.

Assertions Used in Test

9.4.9#6, 9.4.5#10, 9.1.1.6#1, 9.4.5#9, 9.4.5#8, 9.4.5#7

Device States for Test

Configured state only

Overview of Test Steps

The test software performs thefollowing steps:

U2 with SetFeature() to device

1. Send a SetFeature(U2_ENABLE) to the device.
2. Send a GetStatus() to the device and verify that U2 is enabled.
3, Set the U2 Timeout value to 1 for the parent port of the device
4. Allow 520ms for port to go to U2.
5. Bring link back to U0 by changing Link State of parent port to U0
6. Send a ClearFeature(U2_ENABLE) to the device.
7. Set U2 Timeout value to 0 for the parent port of the device.
8. Send a GetStatus() to the device and verify that U2 is disabled.

U1 with SetFeature() to device

9. Send a SetFeature(U1_ENABLE) to the device.
10. Send a GetStatus() to the device and verify that U1 is enabled.
11. Set the U1 Timeout value to 1 for the parent port of the device.
12. Allow 520ms for port to go to U2.
13. Bring link back to U0 by changing Link State of parent port to U0.
14. Send a ClearFeature(U1_ENABLE) to the device.
15. Set U1 Timeout value to 0 for the parent port of the device.
16. Send a GetStatus() to the device and verify that U1 is disabled.

U1 and U2 with SetFeature() to device

17. Send a SetFeature(U1_ENABLE) to the device.
18. Send a GetStatus() to the device and verify that U1 is enabled.
19. Send a SetFeature(U2_ENABLE) to the device.
20. Send a GetStatus() to the device and verify that U2 is enabled.
21. Set U1 and U2 timeout values to 1 for the parent port.
22. Wait 520ms for the port to go to a low power state.
23. Device should go to U1 and then U2.
24. Bring the link back to U0 by setting Port Link State of parent port to U0.
25. Send a ClearFeature(U1_ ENABLE) to the device.
26. Send a GetStatus() to the device and verify that U1 is disabled.
27. Send a ClearFeature(U2_ ENABLE) to the device.
28. Send a GetStatus() to the device and verify that U2 is disabled.

U1 with Force PM Link Accept

29. Enable Force PM Link Accept
30. Set the U1 Timeout value to 1 for the parent port of the device
31. Allow 520ms for the port to go to U2
32. Bring link back to U0 by changing Link State of parent port to U0
33. Set U1 Timeout value to 0 for the parent port of the device.
34. Clear Force PM Link Accept

U1 + U2 with Force PM Link Accept

35. Enable force PM Link Accept
36. Set U1 and U2 timeout values to 1 for the parent port.
37. Wait 520ms for the port to go to a low power state.
38. Device should go to U1 and then U2.
39. Bring the link back to U0 by setting Port Link State of parent port to U0.
40. Disable Force PM Link Accept

44. Re-enumerate the device to make sure the Force PM Link Accept LPM is cleared.

TD.9.25 Deferred Packet Test

This test verifies usage of deferred packets. It is only run behind a SS hub.

Assertions Used in Test

9.4.9#5, 9.4.5#8, 9.4.9#6, 9.4.5#10, 9.1.1.6#1, 9.4.5#7, 9.4.5#9

Device States for Test

Configured state only

Overview of Test Steps

1. Send a SetFeature(U1_ENABLE) to the device
2. Set U1 Timeout value to 1 for the parent port
3. Wait 520ms for link to go to U1
4. Send a GetStatus() to the device. This should cause the hub to generate a deferred packet.
5. The GetStatus() call must succeed.
6. Verify that the link went to U0.
7. Send a ClearFeature(U1_ENABLE) to the device.
8. Set the U1 Timeout value to 0 for the parent port.

Repetitions:

Repeat test steps with only U2 enabled.
Repeat test steps with both U1 and U2 enabled.

TD.9.26 Set Isochronous Delay Test

This test verifies a device’s ability to accept isochronous delay values sent to it

Assertions Used in Test

9.4.11#1, 9.4.11#2, 9.4.11#3

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a SetIsochDelay() with the following values
   - wValue – set to 0x28.
   - wIndex – set to Zero.
   - wLength – set to Zero.

   Test fails if the device responds with a stall or an error.

TD.9.27 Set SEL Test

This test verifies a device’s ability to accept system exit latency values sent to it.

Assertions Used in Test

9.4.12#1, 9.4.12#2

Device States for Test

This test is run with the device in the Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a SetSEL() with the following values

- wValue – set to Zero.
- wIndex – set to Zero.
- wLength – set to Six.
- Data –
  - U1SEL = 0x20
  - U1PEL = 0x0A
  - U2SEL = 0x0815
  - U2PEL = 0x07FF

Test fails if the device responds with a stall or an error.

TD.9.28 SuperSpeedPlus Isochronous Endpoint Companion Descriptor Test

This test verifies that the device under test reports SuperSpeedPlus isochronous endpoint companion descriptors in compliance with the specification.

Assertions Used in Test

9.6.7#20, 9.6.7#22, 9.6.8#1, 9.6.8#2, 9.6.8#3, 9.6.8#4, 9.6.8#5.

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
   - wValue – set to Two (Configuration).
   - wIndex – set to Zero.
   - wLength – wTotalLength (All of the Configuration Descriptor Set).
3. Parse the data returned, keeping all endpoint descriptors obtained
4. Check if each SuperSpeed endpoint has a SuperSpeed Endpoint Companion Descriptor. Test fails if it doesn't: (9.6.7#1)
   - EndpointCompanion.bLength must be equal to six (6d) (9.6.7#3)
   - EndpointCompanion.bDescriptorType must be 48d for SUPERSPEED_USB_ENDPOINT_COMPANION (48d) (9.6.7#4)
5. For each SuperSpeed Endpoint Companion Descriptor, get the following descriptor, if any.
6. If the following descriptor exists, check whether it's a SuperSpeedPlus Isochronous Endpoint Companion descriptor:
   - EndpointCompanion.bLength must be equal to eight (8d) (9.6.8#1)
   - EndpointCompanion.bDescriptorType must be 49d for SUPERSPEEDPLUS_ISOCHRONOUS_ENDPOINT_COMPANION (49d)
7. Check bit 7 of EndpointCompanion.bmAttributes:
   a. If it is one, then test fails if endpoint is Bulk. (9.6.7#9)
   b. If it is one, then test fails if endpoint is Control or Interrupt. (9.6.7#10)
   c. If it is one, and the endpoint is Isochronous, then test fails if following descriptor does not exist, or is not a SuperSpeedPlus Isochronous Endpoint Companion descriptor. (9.6.7#20)
   d. If it is zero, then test fails if following descriptor exists and is a SuperSpeedPlus Isochronous Endpoint Companion descriptor. (9.6.7#20)
8. If there is a valid SuperSpeedPlus Isochronous Endpoint Companion descriptor:
   a. Test fails if the wReserved is not zero (9.6.8#3)
   b. Test fails if dwBytesPerInterval is less than 48K (C000H).(9.6.8#4)

   c. Test fails if dwBytesPerInterval is not less than (MAX_ISO_BYTES_PER_B1_Gen1 x Number of Lanes x LANE_SPEED_MANTISSA_GEN1) / LANE_SPEED_MANTISSA_GEN1. (9.6.8#5)

9. If the device supports multiple configurations, repeat this test for each configuration

TD.9.29 Sublink Speed Test

This test verifies that SuperSpeed devices correctly report sublink speed.

Assertions Used in Test

8.5.6.7#1, 8.5.6.7#3, 8.5.6.7#4, 8.5.6.7#5, 8.5.6.7#6, 8.5.6.7#7, 8.5.6.7#7, 8.5.6.7#9

Device States for Test

This test is run with the device in Default state.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the Default state.
2. Issue SetAddress command.
3. If the DUT is a SuperSpeed Gen1 device:
   a. Test fails if device sent any Device Notification Transaction Packets after the SetAddress (8.5.6.7#1)
4. If the DUT is a SuperSpeed Gen2 device:
   a. Verify that 2 Device Notification Transaction Packets are received after the SetAddress (8.5.6.7#1, 8.5.6.7#5, 8.5.6.7#1, 8.5.6.7#3)
   b. Test fails if any of the following fields are incorrect in the first Dev Notification packet (Rx):
      i. Notification subtype must be 6, Device Notification Transaction Packet (8.5.6.7#1)
      ii. Notification type must be 5, SUBLINK_SPEED (8.5.6.7#1)
      iii. TPF must be 1 (8.5.6.7#4)
      iv. ST must be 0, symmetric (8.5.6.7#6)
      v. Direction must be 0, Rx (8.5.6.7#3)
      vi. LSE must be 3, GBs
      vii. Lane Count must be 0 or 1
      viii. Lane Count in Dev Notification must match Lane Count reported by xHCI/hub.
      ix. LP must be 1 (8.5.6.7#9)
   c. Test fails if any of the following fields are incorrect in the second Dev notification packet (Tx)
      i. Notification subtype must be 6, Device Notification Transaction Packet (8.5.6.7#1)
      ii. Notification type must be 5, SUBLINK_SPEED (8.5.6.7#1)
      iii. TPF must be 0 (8.5.6.7#4)
      iv. ST must be 0, symmetric (8.5.6.7#6)
      v. Direction must be 1, Tx (8.5.6.7#5)
      vi. LSE must be 3, GBs
      vii. Lane Count must be 0 or 1
      viii. Lane Count in Dev Notification must match Lane Count reported by xHCI/hub.
      ix. LP must be 1 (8.5.6.7#9)
   d. Lane Count must be the same for Rx and Tx (8.5.6.7#7)

TD.9.30 Configuration Summary Descriptors Test

This test verifies that the Configuration Summary Descriptor is defined correctly.

Assertions Used in Test

9.6.2.7#1, 9.6.2.7#2, 9.6.2.7#4 – 9.6.2.7#12

Device States for Test

This test is run with the device in Default, Address, and Configured state.

Overview of Test Steps

In this test, the ***signature*** *of a USB Function shall be defined as follows:*

- *a.* *If the USB Function has an Interface Association Descriptor, the signature shall be the combination of the bFunctionClass, bFunctionSubClass and bFunctionProtocol fields of that Interface Association Descriptor.*
- *b.* *If the USB Function does not have an Interface Association Descriptor, the signature shall be the combination of the bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol fields of that Interface Association Descriptor.*

Similarly, the ***signature*** *of a Configuration Summary Descriptor shall be the combination of its bClass, bSubClass, and bProtocol fields.*

In all cases, two signatures shall be equal if the corresponding Class, SubClass and Protocol fields are equal.

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to 1d (DEVICE).
    - wIndex – set to Zero.
    - wLength – 5d.
        - a) Test records an Abort and exits if this does not succeed.
        - b) Save bNumConfigurations for future use.
3. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to 15d (BOS).
    - wIndex – set to Zero.
    - wLength – 5d.
        - a) Test records an Abort and exits if this does not succeed.
4. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to 15d (BOS).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the BOS Descriptor Set).
        - a) Test records an Abort and exits if this does not succeed, or if an unexpected number of bytes are returned.
5. Find and save any Configuration Summary Descriptors contained in the full BOS Descriptor, that is those Descriptors with the following values:
    - wValue – set to 16d (DEVICE CAPABILITY).
    - bDevCapabilityType – set to 16d (CONFIGURATION SUMMARY).
        - a) Test exits if there are no Configuration Summary Descriptors.
        - b) Test records a Failure and exits if bNumConfigurations is set to one and there is at least one Configuration Summary Descriptor. (9.6.2.7#1)
6. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
    - wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – 9d (All of the Configuration Descriptor).
        - a) Test records an Abort and exits if this does not succeed or returns an unexpected number of bytes.
7. Send a GetDescriptor(wValue, wIndex, wLength, Data) request with the following values:
8. wValue – set to Two (Configuration).
    - wIndex – set to Zero.
    - wLength – wTotalLength (All of the Configuration Descriptor Set).
        - a) Test records an Abort and exits if this does not succeed, or if an unexpected number of bytes are returned.
9. Save the results of step 8 for further use.
10. Repeat steps 6-9 for each Configuration in the device, setting wIndex to the appropriate index.
11. Parse each full Configuration Descriptor into USB Functions. *Note: A USB Function is one of the following:*
        - *a.* *An Interface Association (an Interface Association Descriptor, followed by the number of Interfaces given in the bInterfaceCount field.*

    b. *A single Interface that is not a part of an Interface Association.*

12. For each Configuration, save the USB Function signatures for later use.
13. Find all USB Function signatures that do not belong to a USB Function in all Configurations.
    a. Test fails if any of these USB Function signatures are not the signature of a Configuration Summary Descriptor. (9.6.2.7#2)
14. For each Configuration Summary Descriptor:
    a. Test fails if bLength does not equal 9 + bConfigurationCount. (9.6.2.7#4, 9.6.2.7#9)
    b. Test fails if bDescriptorType is not set to 10h. (9.6.2.7#5)
    c. Test fails if bDevCapabilityType is not set to 10h. (9.6.2.7#6)
    d. Test fails if bcdVersion is not set to 0100h. (9.6.2.7#7)
    e. Test fails if the signature is not the signature of a USB Function. (9.6.2.7#8)
    f. Test fails if any bConfigurationIndex is not the index of a Configuration that contains a USB Function with the signature of this Configuration Summary Descriptor (9.6.2.7#10)

Other Tests To Run

1. All Enhanced SuperSpeed devices must demonstrate that they run at all speeds supported as declared in their BOS USB 2.0 Capability Descriptor.
2. All Enhanced SuperSpeed devices must pass class-specific USB tests.

Appendix

Changes in Revision 0.72

1. In TD 9.1, require bcdUSB to be 310H.

Changes in Revision 0.73

1. In TD 9.22, change step 13 to require that the LDM_Enable bit should be set, not cleared.

Changes in Revision 0.74

1. Corrected language in TD 9.20.

Changes in Revision 1.00

1. Editorial fixes.

Changes in Revision 1.01

1. In TD 9.15, handle special case of USB Type-C Bridge devices.

Changes in Revision 1.20

1. Updates for USB 3.2.

    a. In TD 9.1, require bcdUSB to be 0320H.

   b. Allow Lane Count to be 2 (0 for one lane, 1 for two lanes)

Changes in Revision 1.21

1. In TD 9.22, do not run the LDM_ENABLE portions of the test when the Device Under Test is directly connected to a gen 1 speed host port.

Changes in Revision 1.22

1. Fix assertion references for TD 9.14, 9.15, 9.20, 9.22, 9.24, 9.25.
2. Editorial fixes.