

Universal Serial Bus Revision 3.0

USB Command Verifier Compliance Test Specification

Revision 0.9

(DRAFT)

Date: August 17, 2009

Revision: 0.96

Intellectual Property Disclaimer

THIS DOCUMENT IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.

A COPYRIGHT LICENSE IS HEREBY GRANTED TO REPRODUCE AND DISTRIBUTE THIS DOCUMENT FOR INTERNAL USE ONLY. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY OTHER INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.

USB-IF AND THE AUTHORS OF THIS DOCUMENT DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS DOCUMENT. USB-IF AND THE AUTHORS OF THIS DOCUMENT ALSO DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

ALL SUGGESTIONS OR FEEDBACK RELATED TO THIS DOCUMENT BECOME THE PROPERTY OF USB-IF UPON SUBMISSION.

USB-IF MAY MAKE CHANGES TO THIS DOCUMENT, SPECIFICATIONS, PRODUCT DESCRIPTIONS, AND PLANS AT ANY TIME, WITHOUT NOTICE.

Notice: Implementations developed using the information provided in this document may infringe the patent rights of various parties including the parties involved in the development of this document. No license, express or implied, by estoppel or otherwise, to any intellectual property rights (including without limitation rights under any party's patents) are granted herein.

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based solely on these documents but should use the USB IF specifications available through the USB IF. The specifications available through the USB IF have the final authority in any and all cases where a conflict or difference between this specification and the specifications seem to occur.

All product and specification names are trademarks, registered trademarks, or service marks of their respective owners.

Copyright © 2009 USB-IF. All rights reserved.

Revision History

Revision	Issue Date	Comments
0.1	Feb 20, 2008	Initial Version
0.95	June 4, 2009	
0.96	August 17, 2009	

Significant Contributors:

1 Introduction..... 5

2 Assertions and to verify that USB devices are compliant with the USB 3.0 Specification..... 5

3. Test Descriptions for Chapter 9 27

 General Test Initialization 27

 TD.9.1 Device Descriptor Test..... 27

 TD.9.2 Standard Configuration Descriptor Test..... 28

 TD.9.3 Standard Interface Association Descriptor Test 29

 TD.9.4 Standard Interface Descriptor Test..... 30

 TD.9.5 Endpoint Descriptor Test 31

 TD.9.6 SuperSpeed Endpoint Companion Descriptor Test 34

 TD.9.7 BOS and Device Capability Descriptor Test..... 35

 TD.9.8 String Descriptor Test 36

 TD.9.9 Halt Endpoint Test 37

 TD.9.10 Bad Descriptor Test..... 38

 TD.9.11 Bad Feature Test..... 38

TD.9.12 Remote Wakeup Test	39
TD.9.13 Set Configuration Test	39
TD.9.14 Suspend/Resume Test	40
TD.9.15 Function Remote Wakeup Test	40
TD.9.16 Enumeration Test	42
TD.9.17 LTM Test	42
TD.9.18 Bus- or Self- Powered Test	43
TD.9.19 Endpoint Stall Test	44
Other Tests To Run	44

1 Introduction

2 Assertions and to verify that USB devices are compliant with the USB 3.0 Specification.

General Requirements:

- All Reserved fields shall be set to 0.

Assertion #	Assertion Description	Test #
Subsection reference: 4.4.7.2 Interrupt Transfer Bandwidth Requirements		
4.4.7.2#1	A SuperSpeed interrupt endpoint can move up to 3 packets per service interval (MaxBurst <= 2)	9.6
Subsection reference: 9.1.1 USB Device States		
9.1.1 #1	Devices must have a corresponding configuration value for a valid configuration index	9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.9
9.1.1 #2	Devices must support being set to Addressed/Configured state.	9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.9, 9.10, 9.11
9.1.1.3 #1	SuperSpeed capable devices must operate at SuperSpeed when connected to a SuperSpeed capable host.	9.1
9.1.1.3 #2	SuperSpeed devices must support operation at 5GB/s	9.7
9.1.1.3 #3	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 3.0 compliant devices must reset successfully at one of the supported USB 2.0 speeds when in an USB 2.0 only electrical environment.	
9.1.1.4 #1	Devices must use the default address when initially powered or reset.	Test Initialization
9.1.1.5 #1	Devices must default into the fully functional D0 State on initial entry into the configured state	Test Initialization
9.1.1.5 #2	Configuring a device or changing and alternate setting shall reset the status and configurations	
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.	
9.1.1.6 #2	Device shall exit suspend mode when it observes upstream signaling	9.14

Assertion #	Assertion Description	Test #
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	Current Measurement Test Suite
9.2.5.1 #2	A SuperSpeed device operating in SuperSpeed mode shall draw no more than 6 unit loads from Vbus (once it has been configured)	Current Measurement Test Suite
9.2.5.1 #3	A SuperSpeed device operating in a USB 2.0 environment shall draw no more than 500mA from Vbus (once it has been configured)	Current Measurement Test Suite
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.	Current Measurement Test Suite
9.2.5.2 #1	Devices must send a Function Wake Notification after driving resume signaling	9.15
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

Assertion #	Assertion Description	Test #
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.	
9.2.5.4 #3	If a remote wake event occurs in multiple functions, each function shall send a Function Wake	xHCI 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	9.16
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.	
9.2.6.6 #1	Devices capable of operation at SuperSpeed shall be fully functional at one of the USB 2.0 defined speeds.	xHCI Interop Test Procedure
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	9.1
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
9.4 #2	Device shall not set the Halt feature on the control pipe upon receipt of an unsupported or invalid	9.10, 9.11

Assertion #	Assertion Description	Test #
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	9.10
9.4.1#2	Devices must respond to a ClearFeature() request in the addressed or configured state.	9.15
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.	9.17
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.	9.17
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.	9.17
9.4.1#6	Devices must disable/clear the feature in the wValue field in response to the ClearFeature() request.	9.1, 9.11
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
Subsection reference: 9.4.3 Get Descriptor		
9.4.3#1	Devices must respond with a Request Error to GetDescriptor requests that specify an unsupported descriptor type.	9.7
9.4.3#2	Devices must support a valid GetDescriptor(Device) request.	9.5, 9.6
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 Descriptor) in a single request when operating in SuperSpeed mode.	9.6
9.4.3#5	Devices must support a valid GetDescriptor(String) request.	Test Initialization

Assertion #	Assertion Description	Test #
9.4.3#6	Devices must support a valid GetDescriptor(Configuration) request.	Test Initialization
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	A SuperSpeed device shall not report device_qualifier descriptor when operating in SuperSpeed mode	Test Initialization
9.4.3 #11	A SuperSpeed device shall not report other_speed_configuration descriptor when operating in SuperSpeed mode	Test Initialization
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, 9.5
9.4.4#2	A successful GetInterface request must return the alternate setting set by a prior call to SetInterface.	9.4, 9.9
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		
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)	9.15
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() request, must be set to a one.	9.18
9.4.5#5	For a bus-powered device, Bit D0 of status word returned in response to a GetStatus() request, must be set to a zero	9.18

Assertion #	Assertion Description	Test #
9.4.5#6	SuperSpeed Devices must set Bit D1 of Device Status to 0 in response to GetStatus() request	9.14
9.4.5#7	After a successful ClearFeature(U1_Enable) Bit D2 of status word returned in response to a GetStatus request must be set to a zero	9.17
9.4.5#8	After a successful SetFeature(U1_Enable) Bit D2 of status word returned in response to a GetStatus request must be set to a one.	9.17
9.4.5#9	After a successful ClearFeature(U2_Enable) Bit D3 of status word returned in response to a GetStatus request must be set to a zero	9.17
9.4.5#10	After a successful SetFeature(U2_Enable) Bit D3 of status word returned in response to a GetStatus request must be set to a one.	9.17
9.4.5#11	After a successful SetFeature(LTM_ENABLE) Bit D4 of status word returned in response to a GetStatus request must be set to a one.	9.17
9.4.5#12	After a successful ClearFeature(LTM_ENABLE) Bit D4 of status word returned in response to a GetStatus request must be set to a zero.	9.17
9.4.5#13	In response to a GetStatus 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 remote wake up.	9.15
9.4.5#14	After a successful ClearFeature(FUNCTION_SUSPEND) Bit 1 of status word returned in response to a GetStatus 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 status word returned in response to a GetStatus request to the first interface in a function must be set to a one	9.14
9.4.5#16	A GetStatus() request to any interface other than the first interface in a function must return all zeros in the status word.	9.4
9.4.5#17	In response to a GetStatus request to an interrupt or bulk endpoint, if the interrupt or bulk endpoint is currently halted or 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.	9.19

Assertion #	Assertion Description	Test #
9.4.5#18	After the successful completion of a ClearFeature(ENDPOINT_HALT) , SetConfiguration() or SetInterface(), Bit 0 of status word returned in response to a GetStatus() request to an interrupt or bulk endpoint must be set to a zero.	9.19
9.4.5#19	The <i>Self Powered</i> field in the device must not be changed by the SetFeature() or ClearFeature() requests.	
9.4.5#20	In response to a GetStatus() 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() request, must be reset to a zero when the device is reset.	9.17
9.4.5#22	Bit D3 (U2_ENABLE) of the status word returned in response to a GetStatus() request, must be reset to a zero when the device is reset.	9.17
9.4.5#23	Bit D4 (LTM_ENABLE) of the status word returned in response to a GetStatus() request, must be reset to a zero when the device is reset.	9.17
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 Get Status Standard Request	
9.4.5#26	A 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	A 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 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 status word returned in response to a GetStatus request must be set to a one in a SuperSpeed device operating in USB 2.0 mode.	N/A
9.4.5#30	Devices with remote wakeup disabled must not initiate a remote wakeup.	9.15

Assertion #	Assertion Description	Test #
9.4.5#31	Device with remote wakeup enabled must be able to initiate a remote wakeup on it's suspended parent port.	9.15
9.4.5#32	SuperSpeed devices with function remote wakeup disabled must not initiate a function remote wakeup.	9.18
9.4.5#33	SuperSpeed device with function remote wakeup enabled must be able to initiate a function remote wakeupt.	9.18
9.4.5#34	A suspended device must resume normal operation when resume signaling is seen on its upstream port	9.15
9.4.5#35	Bulk and Interrupt endpoints must support ENDPOINT_HALT.	9.9
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.	Test Initialization
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.	
9.4.6#6	Device shall not change its device address until after the Status stage of a SetAddress() request completes successfully	Test Initialization
9.4.7#1	Devices must support a valid SetConfiguration request	Test Initialization
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.	
9.4.7#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.	9.13

Assertion #	Assertion Description	Test #
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
9.4.9#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	
9.4.9#5	Device must not fail a valid SetFeature(U1_Enable) command when in the Configured SuperSpeed state.	9.17
9.4.9#6	Device must not fail a valid SetFeature(U2_Enable) command when in the Configured SuperSpeed state.	9.17
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.	9.17
9.4.9#8	An interface must not fail a valid SetFeature(FUNCTION_SUSPEND) command (in the configured state).	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#10	After a successful SetFeature(SUSPEND), a hub/host port's status must show 'suspended'.	9.15
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, Interop Testing
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, Interop Testing
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	9.14, Interop Testing

Assertion #	Assertion Description	Test #
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	9.14, Interop Testing
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.	9.4
Subsection reference: 9.4.11 Set Isochronous Delay		
Subsection reference: 9.4.12 Set SEL		
Subsection reference: 9.4.13 Sync Frame		
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 Get_Descriptor(Device) request must have the value of 0x03 in the high byte of the bcdUSB field for SuperSpeed devices	9.1
9.6.1#2	The MaxPacketSize of a control endpoint must always be 09H for devices in SuperSpeed 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

Assertion #	Assertion Description	Test #
9.6.1#8	The descriptor returned in response to a GetDescriptor(Device) request must have a length of 0x12 (or appropriate length if less bytes are requested).	9.1
9.6.1#9	The descriptor returned in response to a GetDescriptor(Device) request must return value of 0x01 (DEVICE) in the Descriptor Type field.	9.1
9.6.1#10	The descriptor returned in response to a GetDescriptor(Device) request must return the value of 0x00/0x01/0x10 in the low byte of the bcdUSB field	9.1
9.6.1#11	The descriptor returned in response to a GetDescriptor(Device) request must the value of 0x02 in the high byte of the bcdUSB field for all USB 2.0 speeds	9.1
9.6.1#12	The descriptor returned in response to a GetDescriptor(Device) request must the value of 0x02 in the high byte of the bcdUSB field for high speed devices	Chap 9 Tests for 2.0 devices
9.6.1#13	If serial number is implemented then each instance of a product shall have a unique serial number	
Subsection reference: 9.6.2 Binary Device Object Store (BOS)		
9.6.2#1	All SuperSpeed devices must have a BOS descriptor.	9.7
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 5.	9.7
9.6.2#5	BOS Descriptor field bDescriptorType must be 15	9.7
9.6.2#6	bDeviceCapabilityType field in a Device Capability Descriptor returned in response to a GetDescriptor(BOS) must be 0x02 or 0x03 or 0x04	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	A SuperSpeed device must include the USB 2.0 Extension descriptor and must support LPM when operating in USB 2.0 High-speed mode.	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 10H (DEVICE_CAPABILITY) in the Descriptor Type field.	9.7

Assertion #	Assertion Description	Test #
9.6.2.1#3	The USB 2.0 Extension descriptor returned in response to a GetDescriptor(BOS) request must return value of 02H 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 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 must be set to zero.	9.7
9.6.2.2#1	A 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 10H(DEVICE CAPABILITY) in the Type field.	9.7
9.6.2.2#3	The SuperSpeed USB Device Capability descriptor returned in response to a GetDescriptor(BOS) request must return value of 03H 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.	9.7
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 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.	xHCI Interop Test Procedure
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.	xHCI Interop Test Procedure
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.	xHCI Interop Test Procedure
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 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 must be set to zero.	9.7

Assertion #	Assertion Description	Test #
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	xHCI Interop Test Procedure
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
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 bU2DevExitLatfield	9.7
9.6.2.2#16	Low Speed/Full Speed/High Speed devices must not have a BOS descriptor	N/A
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	9.7
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 10H(DEVICE CAPABILITY) in the Type 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 the container ID	9.7
Subsection reference: 9.6.3 Configuration Descriptor		
9.6.3#1	A device must have at least one configuration.	Test Initialization
9.6.3#2	The descriptor returned in response to a GetDescriptor(Configuration) request must return the value of 0x02 in the Descriptor Type field.	9.2

Assertion #	Assertion Description	Test #
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 the descriptor returned in response to a GetDescriptor(Configuration) request must be set to 1 if the device is self powered	9.2
9.6.3#6	Bit D6 in the attributes field of a the descriptor returned in response to a GetDescriptor(Configuration) request must be set to 0 if the device is bus powered	9.18
9.6.3#7	Bit D5 in the attributes field of a the 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 the descriptor returned in response to a GetDescriptor(Configuration) request must be set to 0 if the device configuration does not support remote wakeup	9.18
9.6.3#9	A BUS POWERED device cannot draw zero power.	Current Measurement Test Suite
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 other speed configuration.	9.2, Also need to update USBCV 2.0 (Device Summary needs to have speeds)
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

Assertion #	Assertion Description	Test #
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 0x07 in the Type 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 power mode cannot draw more than 100ma from the USB bus.	Current Measurement Test Suite
9.6.3#22	A High/Full/Low Speed device operating in Bus powered mode cannot draw more than 500ma.	Current Measurement Test Suite
9.6.3#23	A SuperSpeed device operating in Self powered mode cannot draw more than 150ma from the USB bus.	Current Measurement Test Suite
9.6.3#24	A SuperSpeed device operating in bus powered mode cannot draw more than 900ma.	Current Measurement Test Suite
9.6.3#25	A Device must report the same type of power in the configuration descriptor as in GetStatus.	9.2
9.6.3#26	If a device can continue to operate when disconnected from its external power source it shall continue to do so.	
9.6.3#27	If a device cannot continue to operate when disconnected from its external power source it shall return to the Powered State	xHCI Interop Test Procedure
Subsection reference: 9.6.4 Interface Association Descriptor		
9.6.4#1	The descriptor returned in response to a GetDescriptor(Interface Association) request must return the value of 11 in the Type field.	9.3

Assertion #	Assertion Description	Test #
9.6.4#2	The 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 First Interface Field	9.3
9.6.4#3	The descriptor returned in response to a GetDescriptor(Configuration) request must return the number of contiguous interfaces that are associated with this function in the Interface Count field.	9.3
9.6.4#4	The descriptor returned in response to a GetDescriptor(Configuration) request must not contain zero in the Function class field.	9.3
9.6.4#5	The descriptor returned in response to a GetDescriptor(Configuration) request must contain FFH in the Function class field if the function class is vendor specific.	9.3
9.6.4#6	The length of an Interface Association Descriptor length must be 0x08	9.3
9.6.4#7	The descriptor returned in response to a GetDescriptor(Interface Association) request must not contain zero 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 0x4 in the Descriptor Type field	9.4
9.6.5#2	The descriptor returned in response to a GetDescriptor(Interface) request must return interface number of the interface (which is the zero base 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 descriptor returned in response to a GetDescriptor(Interface) 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 descriptor returned in response to a GetDescriptor(Interface) 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 descriptor returned in response to a GetDescriptor(Interface) request must contain FFH in the Interface class field if the interface class is vendor specific.	9.4

Assertion #	Assertion Description	Test #
9.6.5#6	The descriptor returned in response to a GetDescriptor(Interface) request must contain zero in the Interface sub-class field if the Interface Class field contains zero.	9.4
9.6.5#7	The descriptor returned in response to a GetDescriptor(Interface) 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 descriptor returned in response to a GetDescriptor(Interface) request must contain FFH in the Interface Protocol field if the device uses a vendor specific protocol for this interface.	9.4
9.6.5#9	The descriptor returned in response to a GetDescriptor(Interface) 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.	9.9
9.6.5#13	The interface number cannot be greater than the number of interfaces reported in the configuration descriptor.	9.9
9.6.5#14	An interface descriptor must have exactly the number of endpoint descriptors it specifies in the bNumEndpoints field.	9.4
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.	9.9
Subsection reference: 9.6.6 Endpoint Descriptor		
9.6.6#1	Every endpoint used for an interface must have its own descriptor.	9.9
9.6.6#2	The 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 descriptor returned in response to a GetDescriptor(Endpoint) request must contain 0x5 in the Descriptor Type field	9.5

Assertion #	Assertion Description	Test #
9.6.6#4	Bits 3:0 in the Endpoint address field in the descriptor returned in response to a GetDescriptor(Endpoint) request must contain the endpoint number	9.5
9.6.6#5	Bits 6:4 in the Endpoint address field in the descriptor returned in response to a GetDescriptor(Endpoint) request must be set to zero.	9.5
9.6.6#6	Bit 7 in the Endpoint address field in the descriptor returned in response to a GetDescriptor(Endpoint) request must be zero for an OUT Endpoint, one for an IN Endpoint and must be ignored for control endpoints.	9.5
9.6.6#7	Bits 1:0 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request must be 00 for a Control Transfer.	
9.6.6#8	Bits 1:0 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request must be 01 for a Isochronous Transfer.	
9.6.6#9	Bits 1:0 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request must be 10 for a Bulk Transfer.	
9.6.6#10	Bits 1:0 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are Reserved and must be set to zero for an interrupt endpoint.	9.5
9.6.6#11	Bits 3:2 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are Reserved for an interrupt endpoint.	9.5
9.6.6#12	Bits 5:4 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request must be 00 for Periodic Usage Type for an interrupt endpoint.	9.5
9.6.6#13	Bits 5:4 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request must be 01 for Notification Usage Type for an interrupt endpoint.	9.5
9.6.6#14	Bits 3:2 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are 00 if there is no synchronization for an Isochronous transfer.	
9.6.6#15	Bits 3:2 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are 01 if the synchronization is Asynchronous for an Isochronous transfer.	
9.6.6#16	Bits 3:2 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are 10 if the synchronization is Adaptive for an Isochronous transfer.	

Assertion #	Assertion Description	Test #
9.6.6#17	Bits 3:2 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are 11 if the synchronization is Synchronous for an Isochronous transfer.	
9.6.6#18	Bits 5:4 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are 00 if the Usage Type is Data Endpoint for an Isochronous transfer.	
9.6.6#19	Bits 5:4 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are 01 if the Usage Type is Feedback Endpoint for an Isochronous transfer.	
9.6.6#20	Bits 5:4 in the Attributes field in the descriptor returned in response to a GetDescriptor(Endpoint) request are 10 if the Usage Type is Implicit feedback Data Endpoint for an Isochronous transfer.	
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 descriptor returned in response to a GetDescriptor(Endpoint) request are Reserved and 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 descriptor returned in response to a GetDescriptor(Endpoint) request are Reserved and 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 descriptor returned in response to a GetDescriptor(Endpoint) request must be between 1 to 16 in the Interval field for SuperSpeed isochronous and interrupt endpoints.	9.5
9.6.6#31	The Interval field in the descriptor returned in response to GetDescriptor(Endpoint) is Reserved and must not be used for SuperSpeed Bulk or Control Endpoints.	9.5
9.6.6#32	Only the default control endpoint can have an address of 0x00.	Test Initialization
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 descriptor returned in response to a GetDescriptor(Endpoint) request are set to zero in non-SuperSpeed devices.	Chap 9 Tests for 2.0 devices
9.6.6#35	Legal mult 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-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 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 value is one.	Chap 9 Tests for 2.0 devices
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 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

Assertion #	Assertion Description	Test #
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	A device cannot have a value of 0x00 in the bInterval field.	9.5
9.6.6#51	Bits 4 and 5 cannot BOTH be set in the bmAttributes field.of an Interrupt Endpoint descriptor.	9.5
9.6.6#52	Bits 4 and 5 in the bmAttributes field.of an Interrupt Endpoint descriptor cannot be set to 10	9.5
9.6.6#53	Feedback endpoint must always send data in the opposite direction from the data endpoint it services	9.5
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	9.5
Subsection reference: 9.6.7 SuperSpeed Endpoint Companion Descriptor		
9.6.7#1	All SuperSpeed Devices must have Endpoint Companion Descriptors for each of the endpoints.	9.6
9.6.7#2	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 6 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 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	9.6
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

Assertion #	Assertion Description	Test #
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.	
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 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 if it is a Control or Interrupt Endpoint	9.6
9.6.7#11	Bits 7:2 of the Attributes field in the SuperSpeed Endpoint Companion descriptor returned in response to a GetDescriptor(Configuration) request are Reserved 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 zero if the device supports one Burst of bMaxBurst packets per service interval if it is an Isochronous Endpoint	9.6
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 SuperSpeedCompanionEndpointDescriptors	Chap 9 2.0 Tests
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	xHCI 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	xHCI 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	xHCI Interop Test Procedure

Assertion #	Assertion Description	Test #
9.6.8#1	String descriptors shall all be UNICODE UTF16LE encoded	9.8
9.6.8#2	A device that omits all string descriptors shall not return an array of LANGID codes.	9.8
9.6.8#3	The size of the string descriptor in bytes is computed by subtracting 2 from the value of the first byte of the descriptor	9.8
9.6.8#4	bLength of a UNICODE string descriptor is 2 + length in bytes of the string.	9.8
9.6.8#5	String descriptors are not NULL terminated.	9.8
9.6.8#6	The bDescriptorType of all string descriptors must be 3.	9.8
9.6.8#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

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 is selected.

Assertions Used in Test Initialization

9.2.6.4#1, 9.1.1.4#1, 9.1.1.5#1, 9.2.6.1#1, 9.2.6.3#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

9.1.1#1, 9.1.1#2, 9.6.1#1, 9.6.1#2, 9.6.1#3, 9.6.1#4, 9.6.1#5, 9.6.1#6, 9.6.1#7, 9.6.1#8, 9.6.1#9, 9.6.1#10, 9.6.1#11, 9.1.1.3 #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 Get Descriptor (wValue, wIndex, wLength, Data) request with the following values:
 - wValue – set to One (Device).
 - wIndex – set to Zero.

- wLength – 18d (All of the BOS).

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)

If SuperSpeed Device-

- bcdUSB.hibyte must be 03H
- bcdUSB.lowbyte must be 00H
- bMaxPacketSize0 must be 09H

If High Speed Device –

- bcdUSB.hibyte must be 02H
- bcdUSB.lowbyte must be 10H
- bMaxPacketSize0 must be 40H

If Full Speed Device –

- bcdUSB.hibyte must be 02H
- bcdUSB.lowbyte must be 10H
- bMaxPacketSize0 must be 08H, 10H, 20H, or 40H

If Low Speed Device –

- bcdUSB.hibyte must be 02H
- bcdUSB.lowbyte must be 10H
- bMaxPacketSize0 must be 08H

Check that idVendor is in the list of valid entries maintained by the USB-IF.

If bDeviceClass is 0 then bDeviceSubClass must also be 0

Check that the bDeviceClass is 0 or FFH or is on a list of assigned class codes maintained by the USB-IF.

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

6. Repeat test with the device starting in the following states
 - Default
 - Address
 - Configured
 - Endpoint.MaxPacketSize must be less than or equal to 3584.

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

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

9.6.3#1, 9.6.3#2, 9.6.3#3, 9.6.3#5, 9.6.3#6, 9.4.3#1, 9.4.3#5, 9.4.3#6, 9.4.3#7, 9.4.5#22, 9.4.9#2, 9.4.3#8, 9.6.3#7, 9.6.3#8, 9.6.3#9, 9.6.3#10, 9.6.3#11, 9.6.3#12, 9.6.3#13, 9.6.3#14, 9.6.3#15, 9.6.3#16, 9.6.3#18, 9.6.3#19, 9.6.3#20, 9.6.3#21, 9.6.3#22, 9.6.3#23, 9.6.3#24, 9.4.3#10, 9.4.3#11

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 Get Descriptor (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 Get Descriptor (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).

If SuperSpeed Device –

- Configuration.bMaxPower must be $\leq 112b$ if device is bus powered ($112b * 8mA = 896mA$)
- Configuration.bMaxPower must be $\leq 18b$ if device is self powered ($18b * 8mA = 144mA$)

If Non-SuperSpeed Device –

- Configuration.bMaxPower must be $\leq 250b$ if device is bus powered ($250b * 2mA = 500mA$)
- Configuration.bMaxPower must be $\leq 50b$ if device is self powered ($50b * 2mA = 100mA$)

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. 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

9.4.3#6, 9.4.3#7, 9.4.4#5, 9.6.4#1, 9.6.4#2, 9.6.4#3, 9.6.4#4, 9.6.4#5, 9.6.5#10, 9.6.5#11, 9.6.4#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 Get Descriptor (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 Get Descriptor (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.bFirstInterface must be the Interface Number of the first interface associated with this function as in the Get Descriptor (Interface) request
- InterfaceAssociation.bInterfaceCount must be the number of contiguous interfaces associated with this function
- InterfaceAssociation.bFunctionClass must not be zero and if the function class is vendor specific this field must be FFH. All other values are reserved for assignment by USB-IF
- InterfaceAssociation.bFunctionSubClass must be either FFH or Reserved for assignment by USB-IF

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

6. 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

9.6.3#17, 9.6.5#1, 9.6.5#2, 9.6.5#3, 9.6.5#4, 9.6.5#6, 9.6.5#12, 9.6.5#13, 9.6.5#14, 9.6.5#15, 9.6.5#16, 9.6.5#17, 9.6.5#18, 9.6.6#31, 9.6.6#32, 9.4.4#5, 9.6.5#5, 9.6.5#8, 9.6.5#9, 9.6.5#7, 9.2.3#2, 9.4.5#16, 9.6.5#2, 9.6.5#3, 9.6.5#5, 9.6.5#7, 9.6.5#8, 9.6.5#9, 9.4.10#2

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 Get Descriptor (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 Get Descriptor (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:

- Test fails if a default interface (alternate setting zero) contains an isochronous endpoint with a maximum packet size greater than 0
 - Test fails if a default interface (alternate setting zero) contains an interrupt endpoint with a maximum packet size greater than 64
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 interface number and previous interface number are the same then current alternate setting must be one greater
 - If current interface number and previous interface number 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 greater than 8
 - If Interface.bInterfaceClass is zero then Interface.bInterfaceSubClass must be zero
 - If Interface Class is vendor specific then the Interface Class must be FFH
 - If Interface Class is vendor specific then the Interface Protocol Field must be FFH
 - Verify that descriptor contains the index of string descriptor describing this interface
 - Verify that the Interface Protocol field is 0 for non-class specific protocol
- Test fails if any of the values are not as specified.
9. If the device state is Configured State:
 - Do GetInterface() and verify that the device is initially configured to alternate setting 0.
 - Do SetInterface()
 - Do GetInterface() and verify that the settings are the same
 10. If the device state is in Addressed State:
 - Do GetStatus() call for Interface 1. Device must respond with request error.
 - Do SetInterface(). Device must respond with a request error.
 11. If the test supports multiple configurations repeat test for each configuration descriptor.
 12. If the device is a SuperSpeed device, the test must be run with the device in both SuperSpeed and High Speed.
 13. Repeat test with the device starting in the following states
 - 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#7, 9.6.3#17, 9.6.5#14, 9.6.6#1-9.6.6#50, 9.6.7#1, 9.4.3#2, 9.6.2#28, 9.4.4#1, 9.4.4#2, 9.4.3#1, 9.4.5#23, 9.4.5#22, 9.4.5#24, 9.4.5#25, 9.4.5#26, 9.4.9#13, 9.4.5#27, 9.4.5#28, 9.4.#29, 9.6.7#1, 9.6.7#2, 9.6.6#6, 9.6.6#10, 9.6.6#51, 9.6.6#52, 9.6.6#53, 9.6.6#54

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 Get Descriptor (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 Get Descriptor (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 greater than six (6d)
- Endpoint.bDescriptorType must be five for ENDPOINT (5d)
- Endpoint.bEndPointAddress must not be 0x00 or 0x80
- Endpoint.bmAttributes bits 7 and 6 must be zero
- If SuperSpeed device, each Endpoint must be followed by a SuperSpeed endpoint companion descriptor
- Endpoint.wMaxPacketSize bits 15:13 must be zero for non-SuperSpeed devices
- Endpoint.wMaxPacketSize bits 12:11 must be zero for Low and Full Speed devices
- Devices operating at Low Speeds cannot have Isochronous and Bulk endpoints
- Endpoint.bInterval must not be 0x00

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, if the Endpoint is a Periodic endpoint, bits 5:4 must be 00b
- Endpoint.bmAttributes, if the Endpoint is a Notification endpoint, bits 5:4 must be 01b
- If SuperSpeed device, 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 is zero, then Endpoint.wMaxPacketSize must be between 1 and 1024d.
 - Endpoint.bInterval must be between 1 and 16d.
- Endpoint.bInterval must be greater than 10d for Low Speed devices
- Endpoint.wMaxPacketSize must be between 0 and 1024d for High Speed devices when the Mult value is zero
- Endpoint.wMaxPacketSize must be between 513d and 1024d for High Speed devices when the Mult value is one
- Endpoint.bInterval must be 1 when the Mult value is one
- Endpoint.wMaxPacketSize must be between 683d and 1024d for High Speed devices when the Mult value is two
- Endpoint.bInterval must be 1 when the Mult value is two
- Endpoint.wMaxPacketSize must be 0x08 or less for Low speed devices
- Endpoint.wMaxPacketSize must be 0x40 or less for Full speed devices

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 for SuperSpeed devices
 - Endpoint.wMaxPacketSize must be 0x08 for Low speed devices
 - Endpoint.wMaxPacketSize bits 12:11 must be zero for non-SuperSpeed devices
- If SuperSpeed device, 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 3:2 report Synchronization type
 - If no Synchronization, then bits 3:2 must be 00b
 - If Asynchronous, then bits 3:2 must be 01b
 - If Adaptive, then bits 3:2 must be 10b
 - If Synchronous, then bits 3:2 must be 11b
- Endpoint.bmAttributes, bits 5:4 report Usage type
 - If Data endpoint, then bits 5:4 must be 00b
 - If Feedback endpoint, then bits 5:4 must be 01b
 - If Implicit Feedback Data endpoint, then bits 5:4 must be 10b
 - bits 5:4 must never be 11b
- Endpoint.bInterval must be 0x01 for USB1.x compliant devices
- If SuperSpeed device, each Endpoint must be followed by a SuperSpeed endpoint companion descriptor
 - if EndpointCompanion.bMaxBurst > 0, then Endpoint.wMaxPacketSize must be 1024d
 - if EndpointCompanion.bMaxBurst is zero, then Endpoint.wMaxPacketSize must be between 0 and 1024d.
 - Endpoint.bInterval must be between 1 and 16d.
- Endpoint.wMaxPacketSize must be between 0 and 1023d for Full Speed devices
- Endpoint.wMaxPacketSize must be between 0 and 1024d for High Speed devices when the Mult value is zero
- Endpoint.wMaxPacketSize must be between 513d and 1024d for High Speed devices when the Mult value is one
- Endpoint.bInterval must be 1 when the Mult value is one
- Endpoint.wMaxPacketSize must be between 683d and 1024d for High Speed devices when the Mult value is two
- Endpoint.bInterval must be 1 when the Mult value is two

If Bulk –

- Endpoint.bmAttributes, if set to Control, bits 1:0 must be 10b
- Endpoint.bmAttributes, bits 5:2 must be zero (Reserved)
- Endpoint.wMaxPacketSize must be 1024d
- If SuperSpeed device, Endpoint.bInterval is Reserved and must be zero.
- Endpoint.wMaxPacketSize bits 12:11 must be zero for non-SuperSpeed devices
- Endpoint.wMaxPacketSize must be 0x08/0x10/0x20/0x40 for Full Speed devices
- Endpoint.wMaxPacketSize must be 0x200 for High Speed devices

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

7. Verify that feedback endpoints service endpoints in opposite direction.
8. Verify that explicit feedback endpoints have bits 3:2 of bmAttributes set to 0.
9. If multiple data endpoints are to be serviced by the same feedback endpoint, the data endpoints must have ascending order.

10. Repeat test with the device starting in the following states

- Default
- Address
- Configured state

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

9.6.7#1, 9.6.7#2, 9.6.7#3, 9.6.7#13, 9.6.7#5, 9.6.7#6, 9.6.7#4, 9.6.7#5, 9.6.7#14, 9.6.7#7, 9.6.7#8, 9.6.7#9, 9.6.7#10, 9.6.7#11, 9.6.7#13, 9.6.7#14, 4.4.7.2#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 Get Descriptor (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 greater than six (6d)
 - EndpointCompanion.bDescriptorType must be 48d for SUPERSPEED_USB_ENDPOINT_COMPANION (48d)
 - EndpointCompanion.bMaxBurst must be between 0 and 15d.

If Bulk –

- EndpointCompanion.bmAttributes, bits 4:0 must be between 0 and 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 between 0 and 2d
- EndpointCompanion.bmAttributes, bits 7:2 are reserved and must be 0

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

6. Repeat test with the device starting in the following states
 - Default

- Address
- Configured state

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

9.6.2#1, 9.6.1#2, 9.6.1#3, 9.6.1#4, 9.6.1#5, 9.6.1#6, 9.6.1#7, 9.6.1#8, 9.6.2#12, 9.6.2#23, 9.6.2#24, 9.6.2#25, 9.6.2#26, 9.6.2#14, 9.1.1.3#2, 9.1.1.3#3, 9.6.2#22, 9.6.2#27, 9.6.2#6, 9.6.3#10, 9.6.1#1, 9.6.1#5, 9.6.1#3, 9.6.4#6, 9.6.4#7, 9.4.10#1, 9.6.2#9, 9.6.2#10, 9.6.2#11, 9.6.2#13, 9.6.2#15, 9.6.2#16, 9.6.2#17, 9.6.2#18, 9.6.2#19, 9.6.2#20, 9.6.2#21, 9.4.3#10, 9.6.2.2, 9.6.2.3#1, 9.6.2.3#2, 9.6.2.3#3, 9.6.2.3#4, 9.6.2.3#5, 9.6.2.3#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 Get Descriptor (wValue, wIndex, wLength, Data) request with the following values:
 - wValue – set to 15 (BOS).
 - wIndex – set to Zero.
 - wLength – 5d (All of the BOS).

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

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

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

5. Parse the data returned, only keeping the BOS and Device Capability Descriptors.

Test fails if BOS.bNumDeviceCaps does not reflect the number of Device Capability Descriptors found.

6. Check the returned BOS descriptor for the following values:
 - bLength reflects the size returned.
 - bDescriptorType is 15 (BOS type).

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

7. Check the returned device capability descriptors for the following values:
 - bLength reflects the correct length of the descriptor.
 - bDescriptorType is 02H (USB 2.0 EXTENSION) or 03H (SUPERSPEED_USB) or 04H (CONTAINER_ID)

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 reflects the size of descriptor
 - bDescriptorType is 16 (DEVICE_CAPABILITY)
 - bDevCapabilityType is 02H (USB 2.0 EXTENSION)
 - bmAttributes: Bit 0 is 0

Bit 1 is 1 (for SuperSpeed Devices) or 1 (if Device supports LPM protocol)

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
- bDescriptorType is 16 (DEVICE_CAPABILITY)
- bDevCapabilityType 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

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

6. Repeat test with the device starting in the following states
 - Default
 - Address
 - Configured state

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 if a nonzero string descriptor request is reported

Assertions Used in Test

7.4.5.1.1#1 7.4.5.1.1#2 7.4.5.1.1#4

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 2 (2d)
 - String.bDescriptorType must be three (3d)
 - 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 three (3d)
 - 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. If the device is a SuperSpeed capable device the test must be run with the device in both high speed and SuperSpeed operation.
10. Print out string descriptors, which must be in UNICODE_UTF16LE
11. Repeat test with the device starting in the following states
 - Default
 - Address
 - Configured state

TD.9.9 Halt Endpoint Test

This test is similar to the interface descriptor test except that it also checks that all bulk and interrupt endpoints can be programmatically halted.

Assertions Used in Test

9.4.3#6, 9.6.5#19, 9.6.5#13, 9.4.10#1, 9.6.6#1, 9.4.4#1, 9.4.4#2, 9.6.3#12, 9.4.5#30, 9.6.5#4

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 Get Descriptor (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 Get Descriptor (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:
 - Test fails if a default interface (alternate setting zero) contains an isochronous endpoint with a maximum packet size greater than 0
 - Test fails if a default interface (alternate setting zero) contains an interrupt endpoint with a maximum packet size greater than 64
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 interface number and previous interface number are the same then current alternate setting must be one greater
 - If current interface number and previous interface number 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 greater than 8
 - If Interface.bInterfaceClass is zero then Interface.bInterfaceSubClass must be zero
9. Perform the following additional checks
 - Call set interface for the current bInterfaceNumber and bAlternateSetting values.
 - If the current interface has no alternate settings the device may STALL the request – otherwise it must succeed.
 - Call get interface for the current bInterfaceNumber and verify that the correct alternate setting value is returned.
10. For each endpoint associated with interface descriptor in step 10 perform the following checks if the endpoint is bulk or interrupt:

- Issue a valid Get Status request for the endpoint.
- If the endpoint is halted issue a valid Clear Feature request with feature selector ENDPOINT_HALT. Then issue a valid Get Status request for the endpoint and
- verify that the endpoint no longer reports halt.
- Issue a valid Set Feature request with feature selector ENDPOINT_HALT
- Issue a valid Get Status 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 Get Status request for the endpoint.
- Verify that ENDPOINT_HALT is not reported in the status.

Test fails if any of the above checks fail.

11. If the test supports multiple configurations repeat test for each configuration descriptor.

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#2

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. Place the device in the Address state.
2. Issue a Get Descriptor 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 3.0 specification or any class specifications.
3. Verify that the device responded with a STALL.
4. Issue a Get Descriptor (Device) request to the device, to make sure the device is still responding. If this is successful, skip step 5.
5. Re-enumerate the device.

TD.9.11 Bad Feature Test

This test checks that a device properly handles a Set Feature request when the feature selector is invalid.

Assertions Used in Test

9.4.9#2, 9.1.1#2, 9.4.1#1, 9.4.1#6. 9.4#3

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. Place the device in the Address state.
2. Issue a Set 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 2.0 specification or any class specifications.
3. Verify that the device responded with a STALL.
4. Issue a Get Descriptor (Device) request to the device, to make sure the device is still responding. If this is successful, skip to step 6.

5. Re-enumerate the device. If this fails then the test aborts.
6. 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 2.0 specification or any class specifications.
7. Verify that the device responded with a STALL.
8. Issue a Get Descriptor (Device) request to the device, to make sure the device is still responding. If this is successful, skip step 9.
9. Re-enumerate the device. If this fails then the test aborts.

TD.9.12 Remote Wakeup Test

This test is not used for 3.0 devices operating in Super Speed Mode. See Remote Wakeup test description in USB 2.0 Test Specification.

TD.9.13 Set Configuration Test

This test checks each device configuration can be set using Set Configuration and that the device properly reports its current configuration in response to the Get Configuration command.

Assertions Used in Test

1.1#7, 9.4.7#4, 9.4.7#2, 9.4.7#3, 9.4.7#5, 9.4.2#1,9,2,3#2

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. Place the device in the Addressed state.
2. Issue a GetConfiguration to the device while in the Addressed State.
3. Verify that device returns 0.
2. Send a Get Descriptor (wValue, wIndex, wLength, Data) request with the following values:
 - wValue – set to Two (CONFIGURATION).
 - wIndex – set to Zero.
 - wLength – 9d
3. Issue a valid Set Configuration request using the bConfigurationValue obtained in step two.
Device is now in Configured state
4. Issue a valid Get Configuration request and verify that the configuration set in step 3 is reported.
5. Send a Get Descriptor (wValue, wIndex, wLength, Data) request with the following values:
 - wValue – set to Two (CONFIGURATION).
 - wIndex – set to the index of the configuration to be tested.
 - wLength – 9d
6. Issue a valid Set Configuration request using the bConfigurationValue obtained in step five.
7. Issue a valid Get Configuration request and verify that the configuration set in step 6 is reported.
8. Issue a valid Set Configuration request using configuration value zero.
Device is now in Addressed state.
9. Issue a valid Get Configuration request and verify that the configuration set in step 8 is reported.
10. Issue a Set Configuration request with an invalid configuration Index.
This invalid Set Configuration request should fail.

11. Issue a valid Set Configuration request using the bConfigurationValue obtained in step five.
Device is now in Configured state.
12. Issue a valid Get Configuration request and verify that the configuration set in step 10 is reported.
13. Issue a Set Configuration request with an invalid configuration Index.
This invalid Set Configuration request should fail.

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#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

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. Place the device in the configured state.
2. Issue GetStatus() to device.
3. Verify that bit D1 of status is 0.
4. Suspend the parent port that the device is attached to.
5. Allow 10ms for device to settle.
6. Get Port Status bit state.
7. Port should report that it is suspended.
8. Sleep 200ms
9. Set a GetDescriptor(Device) request to the device. This request should fail because the device should be suspended.
10. Resume the port
11. Send a Get Descriptor (Device) request is sent to the device. If this fails for any reason all devices are re-enumerated to get things back to the initial state. Set SetFeature(FUNCTION_SUSPEND).
12. Send GetStatus request to the first interface.
13. Verify that Bit 1 of Status word is 1.
14. Send ClearFeature(FUNCTION_SUSPEND)
15. Set GetStatus request to first interface.
16. Verify that Bit 1 of Status word is 0.

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

Device States for Test

This test is run with the device in the Configured state. For each remote wakeup capable configuration the test is run with remote wakeup enabled (must work) and disabled (must not work).

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the Address state.
2. Send a Get Descriptor (wValue, wIndex, wLength, Data) request with the following values:
 - wValue – set to Two (Configuration).
 - wIndex – set to Zero.
 - wLength – 9d
3. Perform the following checks on the configuration descriptor value:
 - bLength must 9d
 - wValue – set to Two (Configuration).
 - bmAttributes bit D5 must be 1 (Remote Wakeup)
4. Repeat steps 2 and 3 for each configuration.
5. If bit D5 is never set there is nothing to test.
6. If bit D5 is set for some configurations, issue a valid get configuration for the configuration that was set in Step 1. If bit D5 is set proceed with the Remote Wakeup Test, otherwise there is nothing to test for this configuration.

FUNCTION_REMOTE_WAKEUP_TEST

7. Issue a valid Set Configuration command to set the device into the remote wakeup capable configuration being tested.
8. Issue a valid Get Configuration command and verify the device reports the configuration that was set in step 7.
9. Issue a valid Set Feature command to the first interface with feature selector FUNCTION_SUSPEND and the Low wIndex set to 0x2 for WAKE_ENABLE.
10. Issue a valid Get Status command to the device.
 - Verify that D1 is 0 for a USB 3.0 device
11. Issue a valid Get Status 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.
12. If this is a remote wakeup disabled test:
 - Issue a valid Clear Feature command with feature selector FUNCTION_SUSPEND and wIndex set to 0.
 - Issue a valid Get Status command to the Interface
 - Verify that Function Remote Wakeup is not reported.
12. Suspend the parent port of the device under test.
13. Sleep for 1 second.
14. Check the parent port status to verify that it has suspended.
15. Prompt the user to generate a remote wakeup event on the device under test.
16. Wait for 15 seconds for a DEV_NOTIFICATION events to be generated.
17. 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.
18. 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.
19. 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.
20. If the test supports multiple configurations repeat test for each configuration descriptor.

21. Do GetStatus call to verify that Function Suspend and FunctionRemote Wake settings are identical to original settings.

If this is a Remote Enable Test:

22. Reset Device. This should reset the Function Remote Wake.
23. Issue GetStatus() to device. Remote Wake bit should be 0.
24. Repeat steps 1-15.
25. Wait for 15 seconds.
26. If there is a DEV_NOTIFICATION matching step 17, the test fails.

TD.9.16 Enumeration Test

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

Assertions Used in TestDevice 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:
 - a. Issue a port reset
 - b. Re-enumerate the device tree. If device does not re-enumerate, the test fails.

Tests 9.17 through 9.19 not yet implemented

TD.9.17 LTM Test

This test checks the LTM, U1_Enable and U2_Enable

Assertions Used in Test

9.4.5#7, 9.4.5#8, 9.4.5#9, 9.4.5#33, 9.4.9#6, 9.4.9#8, 9.4.9#9, 9.4.9#10, 9.4.9#11, 9.4.9#12, 9.4.5#21, 9.4.5#22, 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, skip this test.
2. Issue GetStatus to first interface of Device.
 - bRequest =
 - wValue = 0
 - wIndex =
 - wLength = 2
3. Verify that bit D4, LTM Enable, is cleared.
4. Issue SetFeature to device:
 - bRequest =
 - wValue = 50 (LTM_ENABLE)
 - wIndex = 0
 - wLength = 0
5. Issue GetStatus to first interface of Device.
6. Verify that bit D4, LTM_Enable, is set.
7. Wait 5 seconds for DEV_NOTIFICATION to be sent with these values:

-
- 8. Verify that DEV_NOTIFICATION is received in 10 microseconds
- 9. Issue ClearFeature to device:
 - bRequest =
 - wValue = 50 (LTM_ENABLE)
 - wIndex = 0
 - wLength = 0
- 10. Wait 5 seconds. Make sure no LTM DEV_NOTIFICATIONS were received.
- 11. Issue GetStatus to Device.
- 12. Verify that bit D4, LTM Enable, is cleared.
- 13. Deconfigure the device so that it is in addressed state
- 14. Issue SetFeature to enable LTM.
- 15. Verify that SetFeature returns a RequestError.
- 16. Issue a GetStatus to the first interface of the device
- 17. Verify that bit D4, LTM Enable, is clear.
- 18. Restore the device to Configured state.
- 19. Issue GetStatus to Device
- 20. Verify that U1 Enable bit is clear
- 21. Issue SetFeature to Device to Set U1_Enable
 - bRequest=
 - wValue=
 - wIndex=
 - wLength=
- 22. Issue GetStatus
- 23. Verify that U1 Enable bit is set
- 24. Issue ClearFeature to Device to Clear U1_Enable
 - bRequest=
 - wValue=
 - wIndex=
 - wLength=
- 25. Issue GetStatus
- 26. Verify that U1Enable bit is clear
- 27. Repeat steps 20-26 for U2_Enable
- 28. Repeat steps 14-18 for U1_Enable and U2_Enable
- 29. Issue SetFeature() to device to set U1_ENABLE.
- 30. Reset the device
- 31. Issue GetStatus() to device and verify that U1_ENABLE bit is 0
- 32. Issue SetFeature() to device to set U2_ENABLE
- 33. Reset the device
- 34. Issue GetStatus() to device and verify that U2_ENABLE bit is 0
- 35. Issue SetFeature() to device to set LTM_ENABLE
- 36. Reset the device
- 37. Issue GetStatus() to device and verify that LTM_ENABLE bit is 0.

TD.9.18 Bus- or Self- Powered Test

Assertions Used in Test

9.4.5#4, 9.4.5#5, 9.6.3#6

This test checks that device correctly reports bus or self powered.

Device States for Test

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

Overview of Test Steps

1. Issue GetStatus to device. Verify that bit D0 reflects bus or self powered. Prompt user for this.
2. Bit D6 in the attributes field of the Configuration descriptor must match bus/self powered.

TD.9.19 Endpoint Stall Test

Assertions Used in Test

9.4.5#17, 9.4.5#18, 9.4.5#14

This test checks that device correctly reports bus or self powered.

Device States for Test

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

Overview of Test Steps

For Each Bulk and Interrupt Endpoint:

1. Send SetFeature(ENDPOINT_HALT)
2. Send GetStatus to endpoint.
3. Verify that Bit 0 is Set
4. Send ClearFeature(ENDPOINT_HALT) to Endpoint
5. Set GetStatus to endpoint.
6. Verify that Bit 0 is Cleared

Other Tests To Run

1. All SuperSpeed devices must demonstrate that they run at LS/FS/HS if and only if they declare the capability in their descriptor.
2. All SuperSpeed devices must pass class-specific USB tests.