Universal Serial Bus Revision 2.0

USB Command Verifier Compliance Test Specification

Revision 1.42

Date:	June 19, 2019
Revision:	1.42

The information is this document is under review and is subject to change.

Revision History

Revision	Issue Date	Comments
0.5	August 6, 2001	Initial Revision
0.8	November 20, 2001	Updates to Reflect USBCV Beta Release
.9	Dec. 7, 2001	Updates to Reflect Internal Review and Policy Discussions
1.0 RC 1	March. 29 , 2002	Updated to Reflect USBCV RC Development and Policy Decisions / Changes From USB-IF Compliance Workshop Testing
1.1	Sept. 26, 2002	Updated to add a handful of new assertions to address deficiencies identified in 2002 use of USBCV.
1.2	April 10, 2003	Initial draft of USB OTG tests in section 3.6
1.3	December 29, 2015	Added new hub test: Unacknowledged Connect Remote Wake Test.
1.4	December, 22, 2017	Updated Ch 9 Remote Wakeup test to account for device class specific requirements.

Significant Contributors:

Dan S. Froelich	Intel Corporation
Brad Hosler	Intel Corporation
John Howard	Intel Corporation
Rahman Ismail	Intel Corporation
Chris Robinson	Microsoft Corporation

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

1. Introduction

This test specification primarily covers USB-IF testing of devices and hubs for compliance with the standard commands in Chapters 9 and 11 of the USB 2.0 specification. This specification **does not** describe the full set of USB-IF tests and assertions for these devices.

In particular, hubs and devices must also meet the requirements and tests described in the latest versions of the following documents as well as any other tests mandated by the USB-IF:

Universal Serial Bus Implementers Forum Full and Low Speed Compliance Test Procedure

Universal Serial Bus Implementers Forum USB 2.0 Electrical Test Specification

This specification provides a list of test assertions for the Chapter 9 and Chapter 11 descriptors and commands required for all USB low, full, and high speed peripherals and full and high speed hubs. The specification also provides a set of test assertions for command and descriptor rules specific to the HID device class. The assertions provide a partial list of criteria that these device must meet for USB-IF compliance testing. Test descriptions, providing more detailed information on how each of the assertions are tested are also provided in the document.

The test assertions provide a complete list of the requirements that are covered by this specification. The test descriptions can be referenced to obtain specific details on how the assertions will be tested or for more information when the assertions by themselves are unclear.

2. Test Assertions

Note: Test Assertions with Test Descriptions labeled as N/A are either currently not planned for testing in the automated USB-IF test suite. The assertions must still be met.

2.1 Chapter 9

General Chapter 9 Command Assertions

Num **Assertion**

1.1.1 Devices must have a corresponding configuration value for a valid configuration index.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.4.2

Test Description: TD.1.1, TD.1.2, TD.1.3, TD.1.4, TD.1.5, TD.1.6, TD.1.7, TD.1.8 . . .

1.1.2 Devices must support being set to Addressed/Configured state.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.1.1

Test Description: TD.1.1, TD.1.2, TD.1.3, TD.1.4, TD.1.5, TD.1.6, TD.1.7, TD.1.8 . . .

1.1.3 Devices must support a valid GetDescriptor(Device) request.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.4.3

Test Description: TD.1.1

1.1.4 Devices must support a valid GetDescriptor(String) request.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Sections 9.4.3, 9.6.7

Test Description: TD.1.7

1.1.5 Devices must support a valid GetDescriptor(Configuration) request.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.4.3

Test Description: TD.1.3

General Chapter 9 Command Assertions

Num	Assertion
1.1.6	High Speed Capable devices must support a valid GetDescriptor(OtherSpeedConfiguration) request. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.3 **Test Description:** TD.1.3
1.1.7	Devices must support a valid SetConfiguration request **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.7 **Test Description:** TD.1.11
1.1.8	Device must support a valid GetDescriptor(DeviceQualifier) request. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.3 **Test Description:** TD.1.2

HID Command Assertions

Num	Assertion
1.1.9	HID devices must support a GetDescriptor(HIDDescriptor) request. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Sections 6.2, 7.1.1 **Test Description:** TD.3.1
1.1.10	HID devices must support a valid SetProtocol(Report) request. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.2.6 **Test Description:** TD.3.3
1.1.11	HID devices must support a valid GetProtocol request. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.2.5 **Test Description:** TD.3.3
1.1.12	HID devices must support a valid SetProtocol(Boot) request. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.2.6 **Test Description:** TD.3.3
1.1.13	HID devices must support a valid GetDescriptor(Report) request. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Sections 6.2, 7.1.1 **Test Description:** TD.3.4
1.1.14	HID devices that do not support a SetIdle request must respond with a request error. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Sections 7.2, 7.2.4 **Test Description:** TD.3.2
1.1.15	HID devices that do not support a GetIdle request must respond with a request error **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Sections 7.2, 7.2.3 **Test Description:** TD.3.2

General Device Assertions

Num	Assertion
1.1.20	Devices must respond with a Request Error to GetDescriptor requests that specify an invalid descriptor type. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.2.7 and 9.4.3. **Test Description:** TD.1.8
1.1.21	Devices must respond with a Request Error to SetFeature requests that specify an invalid feature selector. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.2.7 and 9.4.9. **Test Description:** TD.1.9
1.1.22	Devices must respond with a Request Error to ClearFeature requests that specify an invalid feature selector. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.2.7 and 9.4.1. **Test Description:** TD.1.9
1.1.30	A suspended device must resume normal operation when resume signaling is seen on its upstream port. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 7.1.7.7. **Test Description:** TD.1.13
1.1.100	Devices must support a Get Status Standard Request **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7. **Test Description:** TD.1.3, TD.1.5
1.1.101	Device Descriptors must use only Class codes defined in the USB specification, or allocated and published by the USBIF. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.1. **Test Description:** TD.1.1

Device Descriptor Assertions

Num	Assertion
1.2.1	The descriptor returned in response to a GetDescriptor(Device) request must have a length of 0x12 (or appropriate length if less bytes are requested). **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.1. **Test Description:** TD.1.1
1.2.2	The descriptor returned in response to a GetDescriptor(Device) request must the have value of 0x01 in the Type field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.1. **Test Description:** TD.1.1
1.2.3	The descriptor returned in response to a GetDescriptor(Device) request must have the value of 0x01/0x02 in the high byte of the bcdUSB field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.1. **Test Description:** TD.1.1
1.2.4	The descriptor returned in response to a GetDescriptor(Device) request must have the value of 0x00/0x01/0x10 in the low byte of the bcdUSB field **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.1. **Test Description:** TD.1.1

Device Descriptor Assertions

Num **Assertion**

1.2.5 The descriptor returned in response to a GetDescriptor(Device) request must have a value of 0x00 in the device subclass field when the device

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.1.

Test Description: TD.1.1

1.2.6 The MaxPacketSize of a control endpoint must be 0x40 for a usb device operating at High speed.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.1.

Test Description: TD.1.1

1.2.7 The MaxPacketSize of a control endpoint must be 0x08 for a usb device operating at Low speed.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.1.

Test Description: TD.1.1

1.2.8 The MaxPacketSize of a control endpoint must be one of 0x08/0x10/0x20/0x40 for a usb device operating at Full speed

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.1.

Test Description: TD.1.1

1.2.9 A device must have at least one configuration.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.3.

Test Description: TD.1.1

1.2.10 The descriptor returned in response to a GetDescriptor(Device) request must the value of 0x02 in the high byte of the bcdUSB field for a high speed device.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.1.

Test Description: TD.1.1

Configuration Descriptor Assertions

Num **Assertion**

1.2.20 The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request cannot contain multiple descriptors of type configuration.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.3

Test Description: TD.1.3

1.2.21 The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration)request cannot contain a descriptor of type device.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.3

Test Description: TD.1.3

1.2.22 The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration)request cannot contain a descriptor of type device qualifier.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.3

Test Description: TD.1.3

1.2.23 The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration)request cannot contain a descriptor of type other speed configuration.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.3

Test Description: TD.1.3

Configuration Descriptor Assertions

Num	**Assertion**
1.2.24	The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request must contain the number of interfaces descriptors reported in the configuration descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.3 **Test Description:** TD.1.3
1.2.25	The descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request must contain the number of endpoint descriptors reported in the contained interface descriptors. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.3
1.2.26	The configuration descriptor returned in response to a GetDescriptor([OtherSpeed]Configuration) request must have a length of 0x09. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.6.3, 9.6.4. **Test Description:** TD.1.3
1.2.27	The descriptor returned in response to a GetDescriptor(Configuration) request must the value of 0x02 in the Type field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.3. **Test Description:** TD.1.3
1.2.28	The descriptor returned in response to a GetDescriptor(OtherSpeedConfiguration)request must the value of 0x07 in the Type field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.4. **Test Description:** TD.1.3
1.2.29	The number of interfaces cannot be a zero in a [OtherSpeed]Configuration descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.6.4, 9.6.5 **Test Description:** TD.1.3
1.2.30	Bits 0 through 4 must be set to zero in the bmAttributes field.of a [OtherSpeed]Configuration descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.6.3, 9.6.4. **Test Description:** TD.1.3
1.2.31	Bit 7 must be set to one in the bmAttributes field of a [OtherSpeed]Configuration descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.6.3, 9.6.4. **Test Description:** TD.1.3
1.2.32	A BUS POWERED device cannot draw zero power. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Sections 9.6.3. **Test Description:** TD.1.3
1.2.33	A SELF POWERED device cannot draw more than 100ma from the USB bus. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 7.2.1.5. **Test Description:** TD.1.3
1.2.34	A BUS POWERED device cannot draw more than 500ma. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 7.2.1.4. **Test Description:** TD.1.3

Interface Descriptor

Num	Assertion
1.2.40	An Isochronous endpoint present in alternate interface 0x00 must have a MaxPacketSize of 0x00 **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 5.6.3. **Test Description:** TD.1.4
1.2.41	The first interface must have an interface number of 0x0 and an alternate setting of 0x0. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.42	The interface number cannot be greater than the number of interfaces reported in the configuration descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.43	An Interface must have at least one setting with an Alternate Setting set to zero. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.44	A device must support the SetInterface request if it has alternate settings for that interface. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.10. **Test Description:** TD.1.4
1.2.45	Alternate settings for a given interface must be in sequential order. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.46	Interface numbers must be in sequential order. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.47	An interface descriptor must have a length of 0x09. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.48	An interface descriptor must have a value of 0x04 in the Type field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.49	An interface descriptor must have a value of 0x00 in the interface subclass field when the interface class is 0x00. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.50	A device must support the GetInterface request if it has alternate settings for that interface. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.4. **Test Description:** TD.1.4
1.2.51	A successful GetInterface request must return the alternate setting set by a prior call to SetInterface. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.4. **Test Description:** TD.1.4

Interface Descriptor

Num	Assertion
1.2.52	A High speed Interrupt endpoint present in alternate interface 0x00 must have a MaxPacketSize <= 0x40. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4
1.2.53	An interface descriptor must have exactly the number of endpoint descriptors it specifies in the bNumEndpoints field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.5. **Test Description:** TD.1.4

EndPoint Descriptor

Num	Assertion
1.2.60	An endpoint descriptor must have a length of 0x07. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.61	An endpoint descriptor must have a value of 0x05 in the Type field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.62	Only the default control endpoint can have an address of 0x00. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.63	Bits 6 and 7 must be set to zero in the bmAttributes field.of an Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 5.3.1.1. **Test Description:** TD.1.5
1.2.64	Bits 4 and 5 cannot BOTH be set in the bmAttributes field.of an Isochronous Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.65	Bits 2 through 5 must be set to zero in the bmAttributes field.of a non Isochronous Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.66	Bits 13 through 15 must be set to zero in the wMaxPacketSize field.of an Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.67	Legal mult values are 0x00/0x01/0x02 for a High speed Isochronous/Interrupt Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5

EndPoint Descriptor

Num	Assertion
1.2.68	Bits 11 and 12 must be set to zero in the wMaxPacketSize field.of a Control/Bulk Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.69	Bits 11 and 12 must be set to zero in the wMaxPacketSize field.of a non High speed Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.70	USB 1.x compliant devices must have a value of 0x01 in the bInterval field of an Isochronous Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.71	USB 2.x compliant devices must have a value between 0x01 and 0x10 in the bInterval field of an Isochronous Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.72	USB 2.x compliant devices must have a value between 0x01 and 0x10 in the bInterval field of a High speed Interrupt Endpoint descriptor **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.73	A device cannot have a value of 0x00 in the bInterval field of a Full/Low speed Interrupt Endpoint descriptor. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5
1.2.74	A device operating at Low speed cannot have Isochronous endpoints. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 5.6.4. **Test Description:** TD.1.5
1.2.75	A Full speed Isochronous endpoint must have a MaxPacketSize between 0 and 1023. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 5.6.3. **Test Description:** TD.1.5
1.2.76	A High speed Interrupt/Isochronous endpoint must have a MaxPacketSize between 0 and 1024 when the Mult value is zero. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 5.6.3. **Test Description:** TD.1.5
1.2.77	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. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.6. **Test Description:** TD.1.5

EndPoint Descriptor

Num **Assertion**

1.2.78 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.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.6.
Test Description: TD.1.5

1.2.79 A device operating at Low speed cannot have Bulk endpoints.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 5.8.3.
Test Description: TD.1.5

1.2.80 A Full speed Bulk endpoint must have a MaxPacketSize of 0x08/0x10/0x20/0x40.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 5.8.3.
Test Description: TD.1.5

1.2.81 A High speed Bulk endpoint must have a MaxPacketSize of 0x200.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 5.8.3.
Test Description: TD.1.5

1.2.82 A Low speed Interrupt endpoint must have a MaxPacketSize less than or equal to 0x08.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 5.7.3.
Test Description: TD.1.5

1.2.83 A Full speed Interrupt endpoint must have a MaxPacketSize less than or equal to 0x40.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 5.7.3.
Test Description: TD.1.5

1.2.84 A Low speed Interrupt endpoint must have a value greater than 10 in the bInterval field.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 5.7.4.

Test Description: TD.1.5

1.2.90 The number of endpoint descriptors in an interface must match the number reported in the interface descriptor.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.5.
Test Description: TD.1.5

1.2.91 A device that has Bulk/Interrupt endpoints must support the Halt Endpoint request on those endpoints.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.4
Test Description: TD.1.6

Interface Descriptor

Num **Assertion**

1.2.100 A device that supports remote wakeup in a specific configuration must not fail a valid SetFeature(DEVICE_REMOTE_WAKEUP) command.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.4.9.

Test Description: TD.1.10

Interface Descriptor

Num	Assertion
1.2.101	A device that supports remote wakeup in a specific configuration must not fail a valid ClearFeature(DEVICE_REMOTE_WAKEUP) command. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.1. **Test Description:** TD.1.10
1.2.102	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 **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.5. **Test Description:** TD.1.9
1.2.103	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. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.5. **Test Description:** TD.1.9
1.2.104	After a successful SetFeature(SUSPEND), a hub/host port's status must show 'suspended'. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.13 **Test Description:** N/A
1.2.105	A device with remote wakeup disabled must not initiate a remote wakeup. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.1. **Test Description:** TD.1.9
1.2.106	A device with remote wakeup enabled must be able to initiate a remote wakeup on it's suspended parent port. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.9. **Test Description:** TD.1.9

Device Qualifier Assertions

Num	Assertion
1.2.120	The descriptor returned in response to a GetDescriptor(DeviceQualifier) request cannot have a value of 0x00/0x01 in the high byte of the bcdUSB field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.2. **Test Description:** TD.1.2
1.2.121	The descriptor returned in response to a GetDescriptor(DeviceQualifier) request must have a length of 0x0a. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.2. **Test Description:** TD.1.2
1.2.122	The descriptor returned in response to a GetDescriptor(DeviceQualifier) request must the value of 0x06 in the Type field. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.2. **Test Description:** TD.1.2
1.2.123	The descriptor returned in response to a GetDescriptor(DeviceQualifier) request must have a value of 0x00 in the device subclass field when the device class is 0x00. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.6.2. **Test Description:** TD.1.2

Device Qualifier Assertions

Num **Assertion**

1.2.124 The descriptor returned in response to a GetDescriptor(DeviceQualifier) request must have a value of 0x00 in the reserved field.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 9.6.2.

Test Description: TD.1.2

2.2 Chapter 11

Hub General Command Assertions

Num **Assertion**

2.1.1 Hubs must support Get_Port_Status requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7.

Test Description: TD.2.1, TD.2.2, TD.2.3, . . . (All 2.x tests)

2.1.2 Hubs must support Clear_Port_Feature(PortPower) requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.2.

Test Description: TD.2.5, TD.2.6

2.1.3 Hubs must support Set_Port_Feature(PortPower) requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.13.

Test Description: TD.2.6

2.1.4 Hubs must support Set_Port_Feature(Suspend) requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.13.

Test Description: TD.2.2, TD.2.5, TD.2.10, TD.2.12

2.1.5 Hubs must support Clear_Port_Feature(Enabled) requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.2

Test Description: TD.2.3, TD.2.9

2.1.6 Hubs must support Set_Feature(RemoteWakeup) requests.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.13

Test Description: TD.2.15, TD.2.16, TD.2.17

2.1.7 Hubs must support Clear_Feature(RemoteWakeup) requests.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.2

Test Description: TD.2.13, TD.2.14

2.1.8 Hubs must support Set_Port_Feature(Reset&Enable) requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.13

Test Description: TD.2.8, TD.2.9, TD.2.10

2.1.9 Hubs must support Clear_Port_Feature(C_PORT_RESET) requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.2

Test Description: TD.2.8, TD.2.9, TD.2.10

2.1.10 Hubs must support Clear_Port_Feature(Suspend) requests on all ports.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.2

Test Description: TD.2.11

Hub General Command Assertions

Num	Assertion
2.1.11	Hubs must support a GetDescriptor(HUB_Descriptor) request. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.5 **Test Description:** TD.2.18

Hub Port Status Change Assertions

Num	Assertion
2.2.1	The state of Hub ports that are suspended, disconnected or enabled must not change when an event (connect, disconnect, suspend, resume, or remote wakeup) happens on another port. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.13, TD.2.14, TD.2.15, TD.2.16, TD.2.17
2.2.2	After receiving a Clear_Port_Feature(PortPower), a hub port's status must show 'not powered'. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.5, TD.2.6
2.2.3	After receiving a Set_Port_Feature(PortPower), a hub port's status must show 'powered'. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2. **Test Description:** TD.2.6
2.2.4	After receiving a Set_Port_Feature(Suspend), a hub port's status must show 'suspended'. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.2, TD.2.5, TD.2.10, TD.2.12
2.2.5	After receiving a Clear_Port_Feature(Enable), a hub port's status must show 'not enabled'. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.3, TD.2.9
2.2.6	After notification that a Reset&Enable request is complete, a hub port's status must show 'enabled'. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.8, TD.2.10
2.2.7	After notification that a Host-Initiated Resume request is complete, a hub port's status must show 'enabled'. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.11
2.2.20	After a Disconnect Event on an enabled port, the hub port status must indicate not Connected, Not Enabled, Not Suspended, Not OverCurrent, Not Reset, and Powered and the port change status must indicate Connect change, No Suspend change, No Overcurrent change, and No Reset change. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.1

Hub Port Status Change Assertions

Num	Assertion
2.2.21	After a Disconnect Event on a suspended port, the hub port status must indicate Not Connected, Not Enabled, Not Suspended, Not OverCurrent, Not Reset, and Powered and the port change status must indicate Connect change, No Suspend change, No Overcurrent change, and No Reset change. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.2
2.2.22	After a Connect Event on a port, the hub port status must indicate. Connected, Not Enabled, Not Suspended, Not OverCurrent, Not Reset, and Powered and the port change status must indicate Connect change, No Enable/Disable change, No Suspend change, No Overcurrent change, and No Reset change. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.4
2.2.23	After a Remote Wakeup Event on a suspended port, the hub port status must indicate Connected, Enabled, Not Suspended, Not OverCurrent, Not Reset, and Powered and the port change status must indicate No Connect change, No Enable/Disable change, Suspend change, No Overcurrent change, and No Reset change. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.24.2.7 **Test Description:** TD.2.7
2.2.40	Hub port events that cause a hub port change bit to transition from a zero to a one (connect, disconnect, reset complete, resume complete) must cause the appropriate notification data to be sent to the host at the next poll of the hub's change notification pipe. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.12.4 **Test Description:** TD.2.1, 2.2. 2.3 (All 2.x tests)
2.2.41	Data returned from a hub's change notification pipe must indicate which ports had a change in status. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.12.4 **Test Description:** TD.2.1, 2.2, 2.3 (All 2.x tests)
2.2.60	A suspended hub that is not enabled for remote wakeup must not signal remote wakeup when a connect or disconnect happens on one of its ports **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 9.4.1 **Test Description:** TD.2.13
2.2.61	A suspended hub that is enabled for remote wakeup must signal remote wakeup when a connect or disconnect happens on one of its ports. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Chapter 11. **Test Description:** TD.2.14
2.2.62	A suspended hub that is enabled for remote wakeup must propogate remote wakeup from downstream ports. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Chapter 11. **Test Description:** TD.2.14

Hub Port Status Change Assertions

Num **Assertion**

2.2.80 User declared inaccessible ports must match hub descriptor device removable list.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Chapter 11.

Test Description: N/A

2.2.90 A hub shall report the PORT_CONNECTION bit in the port status as 1 when the port is in the Powered state, a device attach is detected, and the port transitions from the Disconnected state to the Disabled state.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Chapter 11.

Test Description: TD.2.19

2.2.91 A hub shall report C_PORT_CONNECTION bit as one when the PORT_CONNECTION bit changes because of an attach or detach detect event. Note that this applies even when the device was already connected when the port was being powered on and no reset has occurred on the port.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Chapter 11.

Test Description: TD.2.19

Hub Descriptor Assertions

Num **Assertion**

2.3.1 The descriptor returned in response to a GetDescriptor(Hub_Descriptor) request must have the value of 0x29 in the bDescriptorType field.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.23.2.1.

Test Description: TD.2.18

2.3.2 The descriptor returned in response to a GetDescriptor(Hub_Descriptor) request must have a value of 0x0 or 0x1 in the Logical Power Switching Mode subfield of the wHubCharacteristics field.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.23.2.1.

Test Description: TD.2.18

2.3.3 Self-powered Hubs MUST provide Over-Current Protection.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.23.2.1.

Test Description: TD.2.18

2.3.4 Reserved bits in the Hub Descriptor must read as zeros.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.23.2.1.

Test Description: TD.2.18

2.3.5 Bit 0 of the Device Removable bitmask is reserved and must read as a zero.

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.23.2.1.

Test Description: TD.2.18

2.3.6 All bits in the Hub Descriptors' PortPwrCtrlMask field must be 1's

Specification Ref: Universal Serial Bus Specification, Revision 2.0. Section 11.23.2.1.

Test Description: TD.2.18

Hub Descriptor Assertions

Num	Assertion
2.3.7	The value in the bHubContrCurrent field must not exceed 100mA (0x64) for self-powered hubs. **Specification Ref:** Universal Serial Bus Specification, Revision 2.0. Section 11.23.2.1. **Test Description:** TD.2.18

2.3 HID Specification

General HID Assertions

Num	Assertion
3.2.1	A HID class compliant Interface must have one HID descriptor associated with it. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.1 **Test Description:** TD.3.1
3.2.2	A HID class compliant Interface must have one Endpoint descriptor associated with it. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.1 **Test Description:** TD.3.1

HID Descriptor Assertions

Num	Assertion
3.2.20	A HID class compliant Interface must have a value of 0x00/0x01 in the bInterfaceSubClass field. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section Appendix E.3. **Test Description:** TD.3.1
3.2.21	A HID class compliant Interface must have a value of 0x00/0x01/0x02 in the bInterfaceProtocol field. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section Appendix E.3. **Test Description:** TD.3.1
3.2.22	A HID class compliant Interface must have a valid HID descriptor associated with it. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.1. **Test Description:** TD.3.1
3.2.23	The HID descriptors obtained from the configuration descriptor and one obtained by performing a GetDescriptor request for the same interface must match. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.1. **Test Description:** TD.3.1
3.2.24	The descriptor returned in response to a GetDescriptor(HIDDescriptor) request must have a length of at least 0x09 and in increments of 3 if more than 9. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 6.2.1 **Test Description:** TD.3.1
3.2.25	The descriptor returned in response to a GetDescriptor(HIDDescriptor) request must have a bNumDescriptors value that correlates with the length value. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 6.2.1 **Test Description:** TD.3.1

HID Descriptor Assertions

Num	Assertion
3.2.26	The descriptor returned in response to a GetDescriptor(HIDDescriptor) request must the value of 0x21 in the Type field. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 6.2.1 **Test Description:** TD.3.1
3.2.27	The descriptor returned in response to a GetDescriptor(HIDDescriptor) request must the value greater than 0x01 in the high byte of the bcdHID field. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 6.2.1. **Test Description:** TD.3.1
3.2.28	The descriptor returned in response to a GetDescriptor(HIDDescriptor) request must contain at least one entry containing of type report descriptor. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 6.2.1. **Test Description:** TD.3.1
3.2.29	The descriptor returned in response to a GetDescriptor(HIDDescriptor) cannot contain any optional descriptors with type field set to 0x24-0x2F. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 6.2.1. **Test Description:** TD.3.1

Hid Command Assertions

Num	Assertion
3.2.40	The protocol returned in response to a GetProtocol request must match the protocol set by a successful previous call to SetProtocol. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1. Section 7.2.5 **Test Description:** TD.3.3
3.2.60	The number of items in a report descriptor must be greater than 0x0. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1, Section 6.2.2. **Test Description:** TD..3.4
3.2.61	The report descriptor returned in response to a GetDescriptor(Report) must be compliant with the HID specification. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1, Section 6.2.2. **Test Description:** TD.3.4
3.2.80	The idle rate returned in response to a GetIdle request must match the idle rate set by a successful previous call to SetIdle. **Specification Ref:** Device Class Definition for Human Interface Devices, Revision 1.1, Section 7.2.3. **Test Description:** TD.3.2

3. Test Descriptions

3.1 Chapter 9 (Device) Tests

The Chapter 9 tests cover the device support of the commands set for the in Chapter 9 of the USB specification. There are three different device states. Many of the tests are run with the device under test in more than one of these states. Each of the states and procedure used to put the device into this state are described here. This information is provided to help with debugging in cases where the device under test is not even reaching the desired starting state for the test. The individual tests mention which device states they are run on – but don't repeat the setup procedure. The device states are as follows:

Default State:

1. Put the device in the configured state following the procedure below.
2. Issue a valid Set Configuration command to the device with configuration value zero.
3. Issue a valid Get Configuration command and verify that the device responds with zero.
4. Issue a valid Set Address command to the device with address zero.

Address State:

1. Put the device in the configured state following the procedure below.
2. Issue a valid Set Configuration command to the device with configuration value zero.
3. Issue a valid Get Configuration command to the device and verify that device responds with zero.

Configured State: The device is enumerated using the following procedure.

1. Reset the USB host controller.
2. For each port on the USB host controller turn on port power.
3. Sleep for 1 Second.
4. Drive port reset on each host port. Sleep for 50 milliseconds. Clear port reset.
5. Check host port enable.
6. If host port is enabled enumerate the device on the port.
7. Issue a valid set configuration command for the configuration to be tested.
8. Issue a valid get configuration command and verify that the correct configuration is returned.

Enumerate Device Routine:

1. Get device descriptor using a maximum packet size of 64. (This will cause a short packet for devices with maxpacketsize0 < 64)
2. Issue a Set Address command with the next available USB address.
3. Get the device descriptor again with the right maximum packet size.
4. Get configuration descriptor asking for a number of bytes equal to configuration descriptor size only.
5. Issue a Set Configuration command for the first configuration supported by the device.
6. Get the configuration descriptor again asking for a number of bytes equal to the configuration descriptor size.
7. Get the configuration descriptor using the total length of the entire descriptor.

If the descriptors indicated the device is a hub continue with the steps below:

8. Get the hub class descriptor.
9. Issue a Set Port Feature command with power selector PORT_POWER for each port on the hub.
10. Sleep for 1 second.
11. Issue Get Port Status commands for each port.
12. If the connect bit is set in the port status issue a port reset and proceed to step 13. Otherwise stop.
13. Sleep for 50 milliseconds.
14. Check the port status to ensure the enable bit has been set. Then proceed to the enumerate device routine.

3.1.1 TD 9.1: Device Descriptor Test

This test verifies that the device under test responds to valid Get Device Descriptor commands and returns a descriptor in compliance with the specification.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid get device descriptor command with a requested length of 18 bytes.
3. Perform each of the following checks on the device descriptor value:
   bLength == 18
   bDescriptorType == DEVICE descriptor type
   bcdUSB.hibyte == (01 or 02)
   if (Device Speed == High) then bcdUSB.hibyte == 02
   (if bcdUSB.hibyte == 01) then bcdUSB.lowbyte == (01 or 10 or 00)
   if (bDeviceClass == 0) then bDeviceSubClass == 0.
   if (Device Speed == Low) then bMaxPacketSize0 == 8.
   If (Device Speed == Full) then bMaxPacketSize0 == (8 or 16 or 32 or 64)
   If (Device Speed == High) then bMaxPacketSize0 == 64
   Check that idVendor is in the list of valid entries maintained by the USB-IF.
   bNumConfigurations != 0

   Check that the bDeviceClass == 0 or ==FFH or is on a list of assigned class codes maintained by the USB-IF.
4. Repeat test with device in the default, address, and configured states.
5. If the device supports multiple configurations, the test is repeated for each possible configuration.
6. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

<u>The test *fails* if:</u>

Device enumeration fails following the method described in this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Any of the device descriptor content checks fail.

3.1.2 TD 9.2: Configuration Descriptor Test

This test verifies that the device under test responds to valid Get Configuration Descriptor commands and returns a descriptor in compliance with the specification.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid get configuration descriptor command with a requested length of 9 bytes.
3. Perform the following checks on the configuration descriptor value:
   bLength == 9
   bDescriptorType == CONFIGURATION descriptor type (or OTHER_SPEED_CONFIGURATION type)

4. Issue a valid get configuration descriptor command with a requested length of wTotalLength from the configuration descriptor
5. Parse the data returned with the following information and checks:
   The first descriptor must be a configuration descriptor as checked in step 3.
   All other descriptors must be endpoint, interface, or class descriptors.
   Count the number of endpoint descriptors.
   Count the number of interface descriptors (alternate setting == 0)
   Sum the bNumEndPoint values from all interface descriptors.

6. Check that the number of interface descriptors found equals the bNumInterfaces value in the configuration descriptor.

7. Check that the total number of endpoint descriptors found matches the sum of the bNumEndPoint values from each of the interface descriptors.
8. Issue a valid get configuration descriptor command with a requested length of 9 bytes.
9. Perform each of the following checks on the configuration descriptor value:
   bLength == 9
   bDescriptorType == Configuration descriptor type
   bNumInterface != 0
   bmAttributes bits D0 to D4 == 0.
   bmAttributes bit D7 == 1
   if (bMaxPower == 0) bmAttributes D6 == 1

10. If this is not an other speed configuration test issue a Get Status request for the device.

    If the status request indicates the device is operating bus powered bMaxPower must be less than 500 mA.

    If the status request indicates the device is operating self powered bMaxPower must be less than 100 mA.
11. If the test supports multiple configurations repeat test for each configuration descriptor.
12. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation and is also run for the OTHER speed configuration descriptor.

Results Interpretation

The test transcribes all results to a text based log file.

<u>The test *fails* if:</u>

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

3.1.3 *TD 9.3: Interface Association Descriptor Test*

Defined in the USB 3.2 Test Specification, version 1.22.

3.1.4 TD 9.4: Interface Descriptor Test

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

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.

2. Issue a valid get configuration descriptor command with a requested length of 9 bytes.

3. Perform the following checks on the configuration descriptor value:

```
bLength == 9
bDescriptorType == CONFIGURATION descriptor type (or
 OTHER_SPEED_CONFIGURATION type)
```

4. Issue a valid get configuration descriptor command with a requested length of wTotalLength from the configuration descriptor

5. If this is not an other speed configuration test issue a Set configuration command to set the configuration of the device to the one being tested. Then issue a Get configuration command to make sure the correct configuration is indicated.

6. Perform the following bandwidth check for all interface and endpoint descriptors returned by the device:

If a default interface (alternate setting zero) contains an isochronous endpoint with a maximum packet size greater than 0 the test fails.

If a default interface (alternate setting zero) contains an interrupt endpoint with a maximum packet size greater than 64 the test fails.

7. Check that there is at least one interface descriptor in the data returned.

8. Check that the first interface descriptor returned has bInterfaceNumber and bAlternateSetting values of zero.

9. For each interface descriptor returned perform the following checks:

```
If this is not the first interface descriptor (see check 8)
    If (current interface number  == previous interface number)
        Then current alternate setting must be one greater
    Else
        Current interface number must be 1 greater than previous
        Current alternate setting must be zero
        Current interface number must be < total number interfaces
bLength > 8
bDescriptorType == INTERFACE descriptor type
if (bInterfaceClass == 0) then check that bInterfaceSubClass == 0
bNumEndpoints matches the number of endpoint descriptors.
```

10. If this is not an other speed interface 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.

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

12. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation and is also run for the OTHER speed configuration descriptor.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Valid set interface requests fail for any reason.

Valid get interface requests fail for any reason.

Any of the interface descriptor content checks fail.

3.1.5 TD 9.5: Endpoint Descriptor Test

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

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid get configuration descriptor command with a requested length of 9 bytes.
3. Perform the following checks on the configuration descriptor value:

   bLength == 9
   bDescriptorType == CONFIGURATION descriptor type (or
   OTHER_SPEED_CONFIGURATION type)
4. Issue a valid get configuration descriptor command with a requested length of wTotalLength from the configuration descriptor
5. If this is not an other speed configuration test issue a Set configuration command to set the configuration of the device to the one being tested. Then issue a Get configuration command to make sure the correct configuration is indicated.
6. Issue a valid get device descriptor command to store the bcdUSB value.
7. For each endpoint descriptor found in the data returned from the get configuration descriptor request perform the following checks:

   bLength > 6
   bDescriptorType == ENDPOINT descriptor type
   bEndPointAddress != (0x00 or 0x80)
   bmAttributes D6-D7 must be zero.

   BmAttributes D2-D5 must be zero unless the endpoint is isochronous and bcdUSB is 0200 or greater.

   Note: There is an exception made if bcdUSB is 0x0101 or 0x0110 and the endpoint is part of an audio interface.

   wMaxPacketSize bit D13 to D15 must be zero.

8. The endpoint being tested is categorized according to the following logic:

Current Device Speed	Other Speed Descriptor	Category
Low	NA	CV_LOW
FULL	YES	CV_HIGH
FULL	NO	CV_FULL_2X (bcdUSB)
		CV_FULL_1X
HIGH	NO	CV_HIGH
HIGH	YES	CV_FULL_2X

9. For each endpoint descriptor the following endpoint type and speed dependent checks are performed:

   If wMaxPacketSize bits D11 to D12 are non-zero the endpoint must be type CV_HIGH with transfer type INTERRUPT or ISOCHRONOUS.

10. For Control endpoints the following checks are performed.

CV_LOW	wMaxPacketSize == 8
CV_FULL (1x or 2x)	wMaxPacketSize == (8 or 16 or 32 or 64)
CV_HIGH	wMaxPacketSize == 64

11. For Bulk endpoints the following checks are performed.

CV_LOW	FAIL - ILLEGAL
CV_FULL (1x or 2x)	wMaxPacketSize == (8 or 16 or 32 or 64)
CV_HIGH	wMaxPacketSize == 512

12. For Interrupt endpoints the following checks are performed.

CV_LOW	bInterval >9
	wMaxPacketSize < 9
CV_FULL_1X	bInterval != 0
	wMaxPacketSize < 65
CV_FULL_2X	bInterval != 0
	wMaxPacketSize < 65
CV_HIGH	bInterval > 0 and bInterval < 16
MULT 0	wMaxPacketSize < 1025
MULT 1	wMaxPacketSize > 512 and wMaxSize < 1025
MULT 2	wMaxPacketSize > 682 and wMaxSize < 1025

13. For Isochronous endpoints the following checks are performed.

CV_LOW	FAIL - ILLEGAL
CV_FULL_1X	bInterval == 1
	wMaxPacketSize < 1024
CV_FULL_2X	bInterval > 0 and bInterval < 17
	wMaxPacketSize < 1024
CV_HIGH	bInterval > 0 and bInterval < 17
MULT 0	wMaxPacketSize < 1025
MULT 1	wMaxPacketSize > 512 and wMaxSize < 1025
MULT 2	wMaxPacketSize > 682 and wMaxSize < 1025

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

15. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation and is also run for the OTHER speed configuration descriptor.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Any of the device descriptor content checks fail.

3.1.6 *TD 9.6: Endpoint Companion Descriptor Test*

Not defined for USB 2.0 devices.

3.1.7 TD 9.7: BOS 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.

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. Determine the USB version number of the device:

    - Get the device descriptor.
    - The version number is given in bcdUSB.

Test fails if we cannot get the device's USB version number.

3. 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.

Test fails if the USB version number is less than or equal to 2.00 and the GetDescriptor call succeeds.

4. If the USB version number is less than or equal to 2.00, then exit the test.

Test fails if the USB version number is greater than 2.00 and the GetDescriptor call fails.

5. 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).

6. Check the size of the returned descriptor table.

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

7. Parse the data returned, only keeping the BOS and Device Capability Descriptors.
8. Check the USB version numbers reported in the BOS descriptor.

9. 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 SuperSpeed devices and equal to 3 for USB 3.0 hubs.

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

10. Check the returned device capability descriptors for the following values:.

  - All SuperSpeed devices and USB3.0 Hubs must have a USB 2.0 EXTENSION descriptor.

  - All SuperSpeed devices and USB3.0 Hubs must have a SUPERSPEED_USB descriptor.

  - All USB3.0 Hubs must have a CONTAINER_ID descriptor.

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

11. 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.

12. 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

13. 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

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Any of the values are not as specified.

3.1.8 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 Qualifer or Other Speed descriptor this test is run if a non-zero string descriptor request is reported

Device States For Test

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

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:
   The length must be at least 4. (at least one supported LangID)
   The length must be a multiple of 2.
   The bDescriptorType must be STRING descriptor type.
   The 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:

   The length must be at least 2.

   The bDescriptorType must be STRING descriptor type.

   The number of bytes received must match the bLength field of the desriptor.

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 high speed capable device the test must be run with the device in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Any of the device descriptor content checks fail.

3.1.9 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.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.

2. Issue a valid get configuration descriptor command with a requested length of 9 bytes.

3. Perform the following checks on the configuration descriptor value:
   bLength == 9
   bDescriptorType == CONFIGURATION descriptor type (or OTHER_SPEED_CONFIGURATION type)

4. Issue a valid get configuration descriptor command with a requested length of wTotalLength from the configuration descriptor

5. If this is not an other speed configuration test issue a Set configuration command to set the configuration of the device to the one being tested. Then issue a Get configuration command to make sure the correct configuration is indicated.

6. Perform the following bandwidth check for all interface and endpoint descriptors returned by the device:

If a default interface (alternate setting zero) contains an isochronous endpoint with a maximum packet size greater than 0 the test fails.

If a default interface (alternate setting zero) contains an interrupt endpoint with a maximum packet size greater than 64 the test fails.

7. Check that there is at least one interface descriptor in the data returned.

8. Check that the first interface descriptor returned has bInterfaceNumber and bAlternateSetting values of zero.

9. For each interface descriptor returned perform the following checks:

```
If this is not the first interface descriptor (see check 8)
    If (current interface number  == previous interface number)
        Then current alternate setting must be one greater
    Else
        Current interface number must be 1 greater than previous
        Current alternate setting must be zero
        Current interface number must be < total number interfaces
bLength > 8
bDescriptorType == INTERFACE descriptor type
if (bInterfaceClass == 0) then check that bInterfaceSubClass == 0
```

10. If this is not an other speed interface 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.

11. 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.

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

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Valid set interface requests fail for any reason.

Valid get interface requests fail for any reason.

Valid get status requests fail for any reason.

Valid set feature (ENDPOINT_HALT) requests fail for any reason.

Valid clear feature (ENDPOINT_HALT) requests fail for any reason.

Any of the device descriptor content checks fail.

3.1.10 TD 9.10: Bad Descriptor Test

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

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 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 2.0 specification or any class specifications.
3. Verify that the device responded with a STALL.
4. Issue a GetDescriptor(DeviceDescriptor) request to the device, to make sure the device is still responding. If this is successful, skip step 5.
5. Re-enumerate the device.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

The device cannot be put into the address state.

The device does not respond with a STALL to a GetDescriptor request with an invalid descriptor type.

Valid GetDescriptor requests fail for any reason.

Device enumeration fails.

3.1.11 TD 9.11: Bad Feature Test

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

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 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 2.0 specification or any class specifications.
3. Verify that the device responded with a STALL.
4. Issue a GetDescriptor(DeviceDescriptor) 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 ClearFeature 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 GetDescriptor(DeviceDescriptor) 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.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

The device cannot be put into the address state.

The device does not respond with a STALL to a SetFeature request with an invalid feature selector.

The device does not respond with a STALL to a ClearFeature request with an invalid feature selector.

Valid GetDescriptor requests fail for any reason.

Device enumeration fails.

3.1.12 TD 9.12: Remote Wakeup Test

This test checks each device configuration supporting remote wakeup to ensure that the device properly supports the Set and Clear feature commands with feature selector DEVICE_REMOTE_WAKEUP.

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. Issue a valid get configuration descriptor command with a requested length of 9 bytes.

3. Perform the following checks on the configuration descriptor value:
    bLength == 9
    bDescriptorType == CONFIGURATION descriptor type
    bmAttributes bit D5 == 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.

REMOTE_WAKEUP_TEST

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

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 with feature selector DEVICE_REMOTE_WAKEUP.

10. Issue a valid Get Status command and verify that Remote Wakeup is reported.

11. If this is a remote wakeup disabled test: Issue a valid Clear Feature command with feature selector DEVICE_REMOTE_WAKEUP. Then issue a valid Get Status command verify that 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. Poll the parent port for up to 15 seconds sending valid Get Port Status commands and checking to see when(if) the PORT_SUSPEND bit is cleared.

17. If the test supports multiple configurations repeat test for each configuration descriptor. If the device is HS capable repeat with the device operating at HS and FS.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Valid get status requests fail for any reason.

Valid set feature (DEVICE_REMOTE_WAKEUP) requests fail for any reason.

Valid clear feature (DEVICE_REMOTE_WAKEUP) requests fail for any reason.

Device status does not properly indicate the whether remote wakeup is enabled.

A remote wakeup event is produced by the device when remote wakeup is disabled.

A remote wakeup event is not produced by the device when remote wakeup is enabled.

The device under test prevents its parent port from entering the suspend state.

3.1.13 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..

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 valid get configuration descriptor command with a requested length of 9 bytes for configuration index zero.
3. Issue a valid Set configuration request using the bConfigurationValue obtained in step two.
4. Issue a valid Get configuration request and verify the configuration set in step 3 is reported.
5. Issue a valid get configuration descriptor command with a requested length of 9 bytes for the configuration to be tested.
6. Issue a valid Set configuration request using the bConfigurationValue obtained in step five.
7. Issue a valid Get configuration request and verify the configuration set in step 6 is reported.
8. Issue a valid Set configuration request using configuration value zero.
9. Issue a valid Get Configuration command and verify the device reports the configuration that was set in step 7.
10. Issue a valid Set configuration request using the bConfigurationValue obtained in step five.
11. Issue a valid Get configuration request and verify the configuration set in step 10 is reported.
12. If the test supports multiple configurations repeat test for each configuration descriptor. Also repeat with the device operating at HS and FS if it is HS capable.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Valid get status requests fail for any reason.

Valid set feature (DEVICE_REMOTE_WAKEUP) requests fail for any reason.

Valid clear feature (DEVICE_REMOTE_WAKEUP) requests fail for any reason.

Device status does not properly indicate the whether remote wakeup is enabled.

A remote wakeup event is produced by the device when remote wakeup is disabled.

A remote wakeup event is not produced by the device when remote wakeup is enabled.

The device under test prevents its parent port from entering the suspend state.

3.1.14 TD 9.14: Suspend/Resume Test

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

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. The device is suspended.
2. After waiting ~200ms the device is resumed.
3. After waiting ~200ms a GetDescriptor(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.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

The device is unable to respond to a GetDescriptor(Device) request after being resumed.

3.1.15 *TD 9.15: Function Remote Wakeup Test*

Not defined for USB 2.0 devices.

3.1.16 TD 9.16: Enumeration Test

This test checks that a device can be enumerated multiple times. The device is assigned a different device address in each enumeration cycle. The enumeration method used is the ‘enumerate device routine’ found on page 21.

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. The device is repeatedly enumerated.
3. The full device tree is enumerated to get everything back to a known state.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration, as described in the ‘enumerate device routine’ found on page 21, fails for any reason.

The device does not enumerate at the correct speed during any enumeration.

3.1.17 TD 9.17: Other Speed Configuration Descriptor Test

This test verifies that the device under test responds to valid Get Other Speed Configuration Descriptor commands and returns a descriptor in compliance with the specification.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid get device descriptor command with a requested length of 18 bytes.
3. Perform the following checks on the device descriptor value:
   bcdUSB.hibyte == 02

4. If bcdUSBhibiyte == 02, then perform all the steps of **TD 9.2 Configuration Descriptor Test** where the descriptor type is OTHER_SPEED_DESCRIPTOR_TYPE.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device Descriptor command fails.

. bcdUSB.hibyte != 2.

Any of the failure conditions of **TD 9.2 Configuration Descriptor Test** are reported.

3.1.18 TD 9.18: Device Qualifier Descriptor Test

This test verifies that the device under test responds to valid Get Device Qualifer Descriptor commands and returns a descriptor in compliance with the specification.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid get device qualifier descriptor command with a requested length of 9 bytes.
3. Perform each of the following checks on the device qualifier descriptor value:
   bcdUSB.hibyte == 02
   bLength == 10
   bDescriptorType == DEVICE_QUALIFER descriptor type
   if (bDeviceClass == 0) then bDeviceSubClass == 0.
   If (Device Speed == High) then bMaxPacketSize0 == (8 or 16 or 32 or 64)
   If (Device Speed == Full) then bMaxPacketSize0 == 64
   bNumConfiguration != 0
   bReserved != 0

4. Repeat test with device in the default, address, and configured states.
5. If the test supports multiple configurations repeat test for the configured state for each possible configuration.
6. If the device is a high speed capable device the full test must be run with the device in both full speed and high speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Any of the device descriptor content checks fail.

3.1.19 *TD 9.19: Time Control Transfer Test*

Not defined for USB 2.0 devices.

3.1.20 *TD 9.20: LTM Test*

Not defined for USB 2.0 devices.

3.1.21 TD 9.21: LPM L 1 Suspend Resume Test

This test verifies that if the device under test is required to support LPM, then it supports it correctly.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Determine the USB Version number of the device.
    - Obtain the device descriptor with a GetDescriptor command.
    - The USB version number is contained in the bcdUSB field.
3. If the USB version number is 2.00 or less, then the device is not required to support LPM. End the test.
4. If the device under test is running behind a full speed hub, then end the test.
5. Verify that the parent port of the device under test supports LPM. If it does not, then abort the test.
6. Read the USB 2.0 Extension Descriptor, included in the BOS descriptor.
7. If the BESL and Alternate HIRD Supported bit is not set, this is a failure.
8. If device supports BESL and specifies a Baseline BESL value:
    * Suspend the parent port using the Baseline BESL value.
    * Test fails if device does not go to L1.
    * Resume the port.
    * Test fails if device does not resume to L0.
9. If device supports BESL and specifies a Deep BESL value:
    * Suspend the parent port using the Deep BESL value.
    * Test fails if device does not go to L1.
    * Resume the port.
    * Test fails if device does not resume to L0.
10. For values 0 to 0x0f:
    * Suspend the parent port using the iterator value.
    * Test fails if device does not go to L1.
    * Resume the port.
    * Test fails if device does not resume to L0.

* If device successfully went to L1 for this value, end test.

11. Verify that the device is still alive by retrieving the Device Descriptor.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

The parent port of the device under test does not support LPM.

The call to GetDescriptor in step 8 fails.

3.1.22 *TD 9.22: Set Feature Test*

Not defined for USB 2.0 devices.

3.1.23 *TD 9.23: Reset Device Test*

Not defined for USB 2.0 devices.

3.1.24 *TD 9.24: U1 and U2 Test*

Not defined for USB 2.0 devices.

3.1.25 *TD 9.25: Deferred Packet Test*

Not defined for USB 2.0 devices.

3.1.26 *TD 9.26: Set Isochronous Delay Test*

Not defined for USB 2.0 devices.

3.1.27 *TD 9.27: Set SEL Test*

Not defined for USB 2.0 devices.

3.1.28 *TD 9.28: SuperSpeedPlus Isochronous Endpoint Companion Descriptor Test*

Not defined for USB 2.0 devices.

3.1.29 *TD 9.29: Sublink Speed Test*

Not defined for USB 2.0 devices.

3.1.30 *TD 9.30: Configuration Summary Descriptors Test*

Defined in the USB 3.2 Test Specification, version 1.22, with one modification: if the Device Descriptor field bcdUSB <= 0x0200, then skip the remaining steps.

3.2 Chapter 11 (Hub) Port Tests

All chapter 11 tests that test events that cause port change bits to be set as part of the test attempt to detect the change on the Hub's Interrupt IN status change endpoint. The following procedure is used by all the tests to check the status change endpoint. The individual test descriptions refer to this procedure by name instead of repeating it many times.

STATUS_CHANGE_ENDPOINT_TEST

1. Perform the following steps for up to 15 seconds.
2. Issue interrupt IN commands with a requested length of 8 bytes and a timeout of 50 milliseconds to the Hub Under Tests status change endpoint.
3. Repeat step 2 until the 15 second timeout is exceeded or the interrupt IN command returns data (doesn't time out).
4. If the 15 second timeout is reached the test times out and fails.
5. If data is received from the status change endpoint it must meet the following conditions:
    a. The change bit for the port under test must be set.
    b. NO other change bits for any other ports can be set.
    c. Bit Zero (Hub Change) MUST NOT be set.

NOTE: Not all hub tests involve events that will product a status change. The tests that do, list STATUS_CHANGE_ENDPOINT_TEST as one of their steps.

3.2.1 Detach Device From Enabled Port Test

This test verifies that if a device is detached from an enabled hub port the port status correctly shows a connect change and that port power is still set.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Prompt the user to disconnect the device from the port under test.
6. STATUS_CHANGE_ENDPOINT_TEST
7. Issue a get port status command on the port under test.

8. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE ZERO
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

9. Call Get Port Status on all non test ports.
10. Verify that there were no changes in the port status on any of the non test ports.
11. Repeat steps 1 to 10 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.2 Detach Device From Suspended Port

This test verifies that if a device is detached from a suspended hub port the port status correctly shows a connect change and that port power is still set.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Call Set Port Feature to suspend the port under test.
6. Issue a get port status command to the port under test to verify it is suspended. Continue making calls until suspend is indicated or 1 second elapses.
7. Prompt the user to disconnect the device from the port under test.
8. STATUS_CHANGE_ENDPOINT_TEST
9. Issue a get port status command on the port under test.
10. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE ZERO
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

11. Call Get Port Status on all non test ports.
12. Verify that there were no changes in the port status on any of the non test ports.
13. Repeat steps 1 to 12 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

A valid Set Port Feature command does not suspend the port under test.

3.2.3 Disable The Enabled Port Test

This test verifies that if an enabled port is programmatically disabled no port change bits are set. The test also verifies that the connect and power status bits remain set.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue a Clear Port Feature command with feature selector PORT_ENALBE to disable the port under test.
6. Issue a get port status command on the port under test.
7. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

8. Call Get Port Status on all non test ports.
9. Verify that there were no changes in the port status on any of the non test ports.
10. Repeat steps 1 to 9 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.4 Hot Plug Device Port Test

This test verifies that if a device is connected to a port of the hub under test that status bits in the port are updated appropriately.

Starting Configuration

Port Under Test:	No Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Prompt the user to connect a device to the port under test.
6. STATUS_CHANGE_ENDPOINT_TEST
7. Issue a get port status command on the port under test.
8. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

9. Call Get Port Status on all non test ports.
10. Verify that there were no changes in the port status on any of the non test ports.
11. Repeat steps 1 to 10 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.5 Power Off Suspended Port Test

This test verifies that if a suspended port is powered off the port correctly indicates all zeros for the port status.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Issue a Set Port Feature command with feature selector PORT_SUSPEND.
5. Issue a Get Port Status command on the port under test. Continue making calls until suspend is indicated or 1 second has elapsed.
6. Check that the port status for the port under test is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	IGNORE
PORT_SUSPEND	MUST BE SET
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

7. Issue a Clear Port Feature command with feature selector PORT_POWER.
8. Issue a get port status command on the port under test.

9. Check that the port under test state is as follows:

PORT_CONNECTION	IGNORE
PORT_ENABLE	IGNORE
PORT_SUSPEND	IGNORE
PORT_OVER_CURRENT	IGNORE
PORT_RESET	IGNORE
PORT_POWER	MUST BE ZERO
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	IGNORE
PORT_INDICATOR	IGNORE
C_PORT_CONNECTION	IGNORE
C_PORT_ENABLE	IGNORE
C_PORT_SUSPEND	IGNORE
C_PORT_OVER_CURRENT	IGNORE
C_PORT_RESET	IGNORE

10. Repeat steps 1 to 9 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

The port status for the port under test does not change as described.

3.2.6 Power Off and On Port Test

This test verifies that a hub reports the port status correctly as a port is powered ON and OFF with and without devices connected.

Starting Configuration

Port Under Test:	Any Device/No Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Issue a Clear Port Feature command with feature selector PORT_POWER.
5. Issue a Get Port Status command on the port under test.
6. Check that the port status for the port under test is as follows:

PORT_CONNECTION	IGNORE
PORT_ENABLE	IGNORE
PORT_SUSPEND	IGNORE
PORT_OVER_CURRENT	IGNORE
PORT_RESET	IGNORE
PORT_POWER	MUST BE ZERO

PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	IGNORE
PORT_INDICATOR	IGNORE
C_PORT_CONNECTION	IGNORE
C_PORT_ENABLE	IGNORE
C_PORT_SUSPEND	IGNORE
C_PORT_OVER_CURRENT	IGNORE
C_PORT_RESET	IGNORE

7. Issue a Set Port Feature Command with feature selector PORT_POWER.
8. STATUS_CHANGE_ENDPOINT_TEST
9. Issue a Get Port Status command for the port under test.
10. Check that the port under test state is as follows:

PORT_CONNECTION	SET (if device on test port)
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE (Step 2)
PORT_HIGH_SPEED	MUST NOT CHANGE (Step 2)
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET (if device)
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

11. Repeat steps 1 to 10 with each port as the port under test and with and without a device on the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

The port status for the port under test does not change as described.

3.2.7 Remote Wakeup Port Test

This test verifies that if a device is connected to a port of the hub under test and issues a remote wakeup the port status is updated appropriately.

Starting Configuration

Port Under Test:	Remote Wakeup Capable Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue a Set Port Feature command with feature selector PORT_SUSPEND to suspend the port under test.
6. Issue a get port status command on the port under test. Continue issuing calls until suspend is indicated or 1 second has elapsed.
7. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	IGNORE
PORT_SUSPEND	MUST BE SET
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

8. Prompt the user to issue a remote wakeup on the device on the port under test.
9. STATUS_CHANGE_ENDPOINT_TEST
10. Issue a get port status command on the port under test.

11. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE SET
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE SET
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

12. Verify that there were no changes in the port status on any of the non test ports.
13. Repeat steps 1 to 12 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.8 Reset Enabled Port With Device Connected Port Test

This test verifies that if a device is connected to a port of the hub under test that status bits in the port are updated appropriately when a port reset is performed.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue a Set Port Feature command with feature selector PORT_RESET for the port under test.
6. STATUS_CHANGE_ENDPOINT_TEST
7. Issue a get port status command on the port under test.
8. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE SET
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE SET

9. Call Get Port Status on all non test ports.
10. Verify that there were no changes in the port status on any of the non test ports.
11. Repeat steps 1 to 10 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.9 Reset Disabled Port With Device Connected Port Test

This test verifies that if a device is connected to a disabled port of the hub under test that status bits in the port are updated appropriately when a port reset is performed.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.

2. Check to verify that the ports are in the proper starting configuration using Get Port Status.

3. Clear all port change bits using the Clear Port Feature command.

4. Call Get Port Status on each port and store the initial port status values.

5. Issue a Clear Port Feature command with feature selector PORT_ENABLE for the port under test.

6. Issue a get port status command on the port under test.

7. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

8. Issue a Set Port Feature command with feature selector PORT_RESET for the port under test.

9. STATUS_CHANGE_ENDPOINT_TEST

10. Issue a get port status command on the port under test.

11. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE SET
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE SET

12. Call Get Port Status on all non test ports.
13. Verify that there were no changes in the port status on any of the non test ports.
14. Repeat steps 1 to 13 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.10 Reset Suspended Port Test

This test verifies that if a device is connected to a suspended port of the hub under test that status bits in the port are updated appropriately when a port reset is performed.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue a Set Port Feature command with feature selector PORT_SUSPEND for the port under test.
6. Issue a get port status command on the port under test. Continue to send Get Port Status commands until suspend is indicated or 1 second has elapsed.

7. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	IGNORE
PORT_SUSPEND	MUST BE SET
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

8. Issue a Set Port Feature command with feature selector PORT_RESET for the port under test.

10. STATUS_CHANGE_ENDPOINT_TEST

11. Issue a get port status command on the port under test.

12. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE SET
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE SET

13. Call Get Port Status on all non test ports.

14. Verify that there were no changes in the port status on any of the non test ports.

15. Repeat steps 1 to 13 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.11 Resume Port Test

This test verifies that if a device is connected to a suspended port of the hub under test that status bits in the port are updated appropriately when a port resume is performed.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.

2. Check to verify that the ports are in the proper starting configuration using Get Port Status.

3. Clear all port change bits using the Clear Port Feature command.

4. Call Get Port Status on each port and store the initial port status values.

5. Issue a Set Port Feature command with feature selector PORT_SUSPEND for the port under test.

6. Issue a get port status command on the port under test. Continue get port status commands until suspend is indicated or 1 second has elapsed.

7. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	IGNORE
PORT_SUSPEND	MUST BE SET
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

8. Delay for 1 second to ensure suspend has completed.

9. Issue a Clear Port Feature command with feature selector PORT_SUSPEND for the port under test.

10. STATUS_CHANGE_ENDPOINT_TEST

11. Issue a get port status command on the port under test. Continue get port status commands until suspend is cleared or 1 second has passed.

12. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE SET
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE SET
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

13. Call Get Port Status on all non test ports.
14. Verify that there were no changes in the port status on any of the non test ports.
15. Repeat steps 1 to 14 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

<u>The test *fails* if:</u>

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.2.12 Suspend Port Test

This test verifies that a hub correctly responds to a suspend port command.

Starting Configuration

<u>Port Under Test:</u>	Any Device	Port Enabled
<u>Other Ports:</u>	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue a Set Port Feature command with feature selector PORT_SUSPEND for the port under test.
6. Issue a get port status command on the port under test. Continue get port status commands until suspend is indicated or 1 second has elapsed.

7. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	IGNORE
PORT_SUSPEND	MUST BE SET
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

8. Call Get Port Status on all non test ports.
9. Verify that there were no changes in the port status on any of the non test ports.
10. Repeat steps 1 to 9 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

3.3 Chapter 11 (Hub) Global Tests

Hub global tests check the behavior of the hub while it is suspended and events occur on its downstream ports.

3.2.13 Unacknowledged Remote Wake Connect Test.

This test verifies that a hub correctly responds to a suspend port command.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Prompt user to attach device to port under test.
3. Issue a get port status command on the port under test. Continue get port status commands until suspend is indicated or 5 seconds have elapsed.

4. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	IGNORE
PORT_SUSPEND	IGNORE
PORT_OVER_CURRENT	IGNORE
PORT_RESET	IGNORE
PORT_POWER	IGNORE
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	IGNORE
PORT_INDICATOR	IGNORE
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	IGNORE
C_PORT_SUSPEND	IGNORE
C_PORT_OVER_CURRENT	IGNORE
C_PORT_RESET	IGNORE

5. Call Get Port Status on all non test ports.
6. Verify that there were no changes in the port status on any of the non-test ports.
7. Enable Remote Wake on Hub Under Test.
8. Suspend the parent port of Hub Under Test.
9. Verify that a Remote Wake is generated. (Suspend bit is cleared on port under test).
10. Repeat steps 1 to 9 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non-test ports during the course of the test.

The port status for the port under test does not change as described.

A remote wake is not generated on port under test.

3.2.14 Suspend and Ignore Connect

This test disables remote wakeup for the hub under test and verifies that the hub correctly ignores connect events when it is suspended.

Starting Configuration

Port Under Test:	No Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue the Clear Feature command with feature selector DEVICE_REMOTE_WAKEUP to disable remote wakeup for the hub under test.
6. Suspend the parent port for the hub under test.
7. Get the parent port status to verify the suspend occurred. Continue get parent port status commands until suspend is indicated or 1 second has elapsed.
8. Prompt the user to connect a device to the port under test.
9. Verify the parent port of the hub under test is still suspended.
10. Issue a clear port feature with feature selector PORT_SUSPEND to the parent port of the hub under test.
11. Read the status change endpoint of the hub under test.
12. Verify that the hub reports a status change only on the port under test, and any other ports with devices attached.
13. Issue a get port status command for the port under test.
14. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

15. Call Get Port Status on all non test ports.
16. Verify that there were no changes in the port status on any of the non test ports.

17. Repeat steps 1 to 16 with each port as the port under test. Also repeat with the NON-TEST ports initially having a device connected (GROUP).

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non-test ports which do not have devices attached during the course of the test.

The port status for the port under test does not change as described.

A valid Clear Feature command is not accepted.

3.2.15 Suspend and Ignore Disconnect

This test disables remote wakeup for the hub under test and verifies that the hub correctly ignores disconnect events when it is suspended.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue the Clear Feature command with feature selector DEVICE_REMOTE_WAKEUP to disable remote wakeup for the hub under test.
6. Suspend the parent port for the hub under test.
7. Get the parent port status to verify the suspend occurred. Continue get parent port status commands until suspend is indicated or 1 second has elapsed.
8. Prompt the user to disconnect the device to the port under test.
9. Check that the parent port of the hub under test is still suspended.
10. Issue a clear port feature with feature selector PORT_SUSPEND to the parent port of the hub under test.
11. Read the status change endpoint of the hub under test.
12. Verify that the hub reports a status change only on the port under test, and any other ports that have devices attached.
13. Issue a get port status command for the port under test.

14. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE ZERO
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

15. Call Get Port Status on all non test ports.

16. Verify that there were no changes in the port status on any of the non test ports.

17. Repeat steps 1 to 16 with each port as the port under test. Also repeat with the NON-TEST ports initially having a device connected (GROUP).

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non-test ports during the course of the test which do not have devices attached.

The port status for the port under test does not change as described.

A valid Clear Feature command is not accepted.

3.2.16 Suspend and Disconnect Device

This test suspends the hub under test and verifies that a remote wakeup occurs when a device is disconnected from the hub under test.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.

2. Check to verify that the ports are in the proper starting configuration using Get Port Status.

3. Clear all port change bits using the Clear Port Feature command.

4. Call Get Port Status on each port and store the initial port status values.

5. Issue the Set Feature command with feature selector DEVICE_REMOTE_WAKEUP to enable remote wakeup for the hub under test.

6. Suspend the parent port for the hub under test.

7. Get the parent port status to verify the suspend occurred. Continue get parent port status commands until suspend is indicated or 1 second has elapsed.

8. Prompt the user to disconnect the device to the port under test.

9. Poll the status of the parent port of the hub under test until it is no longer suspended.

10. STATUS_CHANGE_ENDPOINT_TEST

11. Issue a get port status command for the port under test.

12. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE ZERO
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

13. Call Get Port Status on all non test ports.

14. Verify that there were no changes in the port status on any of the non test ports.

15. Repeat steps 1 to 14 with each port as the port under test. Also repeat with the NON-TEST ports initially having a device connected (GROUP).

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

A valid Clear Feature command is not accepted.

3.2.17 Suspend and Connect Device

This test suspends the hub under test and verifies that a remote wakeup occurs when a device is connected to the hub under test.

Starting Configuration

Port Under Test:	Any Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.

2. Check to verify that the ports are in the proper starting configuration using Get Port Status.
3. Clear all port change bits using the Clear Port Feature command.
4. Call Get Port Status on each port and store the initial port status values.
5. Issue the Set Feature command with feature selector DEVICE_REMOTE_WAKEUP to enable remote wakeup for the hub under test.
6. Suspend the parent port for the hub under test.
7. Get the parent port status to verify the suspend occurred. Continue get parent port status commands until suspend is indicated or until 1 second has elapsed.
8. Prompt the user to connect the device to the port under test.
9. Poll the status of the parent port of the hub under test until it is no longer suspended.
10. STATUS_CHANGE_ENDPOINT_TEST
11. Verify that the hub reports a status change only on the port under test.
12. Issue a get port status command for the port under test.
13. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE ZERO
PORT_SUSPEND	MUST BE ZEO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	IGNORE
PORT_HIGH_SPEED	IGNORE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE SET
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

14. Call Get Port Status on all non test ports.
15. Verify that there were no changes in the port status on any of the non test ports.
16. Repeat steps 1 to 15 with each port as the port under test. Also repeat with the NON-TEST ports initially having a device connected (GROUP).

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

A valid Clear Feature command is not accepted.

A remote wakeup is not produced by the hub under test in response to a connect event.

3.2.18 Suspend and Remote Wakeup

This test suspends the hub under test and verifies that a remote wakeup occurs when a remote wakeup is initiated from a downstream device connected to the hub.

Starting Configuration

Port Under Test:	Remote Wakeup Capable Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.

2. Check to verify that the ports are in the proper starting configuration using Get Port Status.

3. Clear all port change bits using the Clear Port Feature command.

4. Call Get Port Status on each port and store the initial port status values.

5. Issue the Set Feature command with feature selector DEVICE_REMOTE_WAKEUP to enable remote wakeup for the hub under test.

6. Suspend the parent port for the hub under test.

7. Get the parent port status to verify the suspend occurred. Continue get parent port status commands until suspend is indicated or until 1 second has elapsed.

8. Prompt the user to initiate a remote wakeup from the device on the port under test.

9. Poll the status of the parent port of the hub under test until it is no longer suspended.

10. STATUS_CHANGE_ENDPOINT_TEST

11. Issue a get port status command for the port under test.

12. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE SET
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE ZERO
C_PORT_ENABLE	MUST BE ZERO
C_PORT_SUSPEND	MUST BE SET
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	MUST BE ZERO

13. Call Get Port Status on all non test ports.

14. Verify that there were no changes in the port status on any of the non test ports.

15. Repeat steps 1 to 14 with each port as the port under test. Also repeat with the NON-TEST ports initially having a device connected (GROUP).

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Clear Port Feature commands fail for any reason.

A known good device fails to enumerate for any reason.

Any port status bits change on the non test ports during the course of the test.

The port status for the port under test does not change as described.

A valid Clear Feature command is not accepted.

A remote wakeup is not produced by the hub under test in response to a connect event.

3.2.19 Suspend and Hub Remote Wakeup on Unacknowledged Connection

This test attaches a device downstream on a downstream port but does not acknowledge the connection, then suspends the hub under test and verifies that a remote wakeup occurs.

Starting Configuration

Port Under Test:	No Device	Port Enabled
Other Ports:	No Device	Port Enabled

Overview of Test Steps

The test software performs the following steps.

1. Enumerate the USB bus.
2. Prompt user to detach the hub under test.
3. Prompt user to power cycle the hub under test.
4. Prompt user to attach a device on the port under test.
5. Prompt user to reattach the hub under test to the same port.
6. Configure the hub under test.
7. Check that the port under test state is as follows:

PORT_CONNECTION	MUST BE SET
PORT_ENABLE	MUST BE SET
PORT_SUSPEND	MUST BE ZERO
PORT_OVER_CURRENT	MUST BE ZERO
PORT_RESET	MUST BE ZERO
PORT_POWER	MUST BE SET
PORT_LOW_SPEED	MUST NOT CHANGE
PORT_HIGH_SPEED	MUST NOT CHANGE
PORT_TEST	MUST BE ZERO
PORT_INDICATOR	MUST BE ZERO
C_PORT_CONNECTION	MUST BE_SET
C_PORT_ENABLE	IGNORE
C_PORT_SUSPEND	MUST BE ZERO
C_PORT_OVER_CURRENT	MUST BE ZERO
C_PORT_RESET	IGNORE

8. Suspend the parent port of the hub under test.
9. Verify that the hub under test generates a remote wake.
10. Repeat steps 1 to 9 with each port as the port under test.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Valid Get Port Status commands fail for any reason.

Valid Set Port Feature.commands fail for any reason.

The port status for the port under test does not change as described.

A remote wakeup is not produced by the hub under after it is suspended.

3.4 Chapter 11 Hub Descriptor Test

3.2.18 Hub Descriptor Test

This test verifies that the device under test responds to valid Get Hub Descriptor command and returns a descriptor in compliance with the specification.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid Get Configuration descriptor command with a requested length of 9 bytes.
2. Issue a valid Set Configuration command with the bConfiguration value from the configuration descriptor.
3. Issue a valid Get Hub Descriptor command with a requested length of 9 bytes.
4. Perform each of the following checks on the device descriptor value:
    bLength == 9 (Note: if the bNbrPorts field is > 7 the length must be longer)

    if bLength > 9 the Get Hub Descriptor is issued again with the indicated length.

    bDescriptorType == HUB descriptor type

    if wHubCharacteristics D2 == 1 then the Device Removable bitmask must have at least one bit set.

    wHubCharacterstics D0 to D1 == (00 or 01)
    wHubCharacteristics D3 to D4 == (00 or 01)

    If wHubCharacteristics D3 to D4 != (00 or 01) issue a Get Status command to the hub. The status must indicate bus powered.
    If bcdUSB.hibyte < 2 then wHubCharateristics D5 to D7 == 0
    wHubCharacteristics D8 t o D15 == 0

    Bit zero in the DeivceRemovableBitMask must be zero. The remaining bits are checked against the user supplied information on non-accessible ports.
    All bits in the PortPwrCtrlMask must be one.
    The bHubContrCurrent must be <= 100 mA for self-powered hubs.
5. If the hub supports multiple configurations repeat test for the configured state for

each possible configuration.

6. If the hub is a high speed capable device the test must be run with the hub in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Any of the hub descriptor content checks fail.

3.5 HID Tests

The following set of HID device class tests share a common algorithm to verify that the device under test is a HID class device. The algorithm is listed below. It is used by USBCV to determine whether the HID tests should be run for the device under test.

HID Class Algorithm:

1. Issue a valid Get Device Descriptor command to the device.
2. For each configuration that the device supports issue a valid get configuration descriptor call with a requested length of 9. Next issue a valid get configuration command with a request length of the wTotalLength field of the configuration descriptor.
3. For each configuration descriptor obtained in step 2 parse the descriptor finding all interface descriptors. If any of the interface descriptors for any of the configurations have a bDeviceClass of 0x03 (HID_CLASS) the device is a HID device and tests below will be run.

3.5.1 HID Descriptor Test

This test verifies that the device under test responds to valid Get HID Descriptor command and returns a descriptor in compliance with the HID specification. It also verifies that the HID descriptor(s) in the configuration descriptor and the descriptor(s) returned explicitly in response to the Get HID Descriptor command are identical.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid Set Configuration command for the configuration to be tested (unless it is an other speed configuration descriptor).

2. Issue a valid Get Configuration command and verify that the configuration has been set. (if we are not testing an other speed configuration descriptor)

3. Issue a valid Get Configuration Descriptor with a requested length of 9 bytes.

4. Issue a valid Get Configuration Descriptor with a requested length of the wTotalLength field from the descriptor obtained in step 3.

5. For the configuration descriptor obtained in step 4 parse the descriptor to locate all interface descriptors.

6. For each interface descriptor check to see if bInterfaceClass == 0x03 (HID_CLASS)
7. Perform each of the following checks on the HID class interfaces:
   bInterfaceSubClass < 2 //Undefined SubClass
   bInterfaceProtocol < 3 //Undefined
8. For each HID interface found in step 6 locate the HID descriptor as follows:
   Issue a valid Get Configuration Descriptor call requesting 9 bytes.
   Issue a valid Get Configuration Descriptor call requesting the full length.

   Look at each interface descriptor in order until the interface and alternate setting numbers match the interface found in step 6.

   Check that bInterfaceClass == 0x03.

   Search forward in the configuration descriptor from this interface descriptor until a HID class descriptor is found (If none are found the test fails).

9. If the HID class descriptor found in 8 is for an interface with alternate setting zero and the configuration descriptor being tested is NOT an other speed descriptor then a valid Get HID Descriptor is issued for the interface in question.

10. The Hid Class descriptor(s) obtained in step 8 and step 9 are compared to ensure that they are identical (It is a failure if they different).

11. For each HID class descriptor obtained in step 9 (and 10) perform the following checks:

    bLength > 8.

    bLength must be a multiple of 3.

    bLength == (9 + (bNumDescriptors –1)*3)

    bDescriptorType == HID descriptor type (0x21)

    bcdHID >= 0x100

    Each Additional bDescriptorType field must not be in the reserved range from 0x24 to 0x2F.

12. The test is repeated for each possible device configuration. For HS capable device it is run with the device operating at both FS and HS. The test is also performed on the other speed configuration descriptor as described above.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Any of the HID descriptor content checks fail.

Valid get HID descriptor commands fail for any reason.

3.5.2 HID Get/Set Idle Test

This test verifies that the device under test correctly supports the Get Idle command if it supports the Set Idle command.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid Set Configuration command for the configuration to be tested (unless it is an other speed configuration descriptor).

2. Issue a valid Get Configuration command and verify that the configuration has been set. (if we are not testing an other speed configuration descriptor)

3. Issue a valid Get Configuration Descriptor with a requested length of 9 bytes.

4. Issue a valid Get Configuration Descriptor with a requested length of the wTotalLength field from the descriptor obtained in step 3.

5. For the configuration descriptor obtained in step 4 parse the descriptor to locate all interface descriptors.

6. For each interface descriptor check to see if bInterfaceClass == 0x03 (HID_CLASS)

7. For each interface found in Step 6 the following descriptor parsing is performed:

REPORT DESCRIPTOR PARSING

8. Issue a valid Get HID descriptor for the interface being tested.
9. Parse the HID descriptor to find the length of the report descriptor for this interface.
10. Issue a valid Get Report Descriptor request with a requested length from step 9.
11. Count the number of item tags in the Report Descriptor. It must be at least 1.
12. Count the number of input tags in the Report Descriptor.
13. Count the number of report Ids in the Report Descriptor.
14. If there are no report Ids the GET/SET Idle test is run with value 0x7F

    If there are report Ids the Get/Set Idle test is run with values 0x00, 0x7F, and 0xFF for each report Id for an INPUT.

GET/SET IDLE TEST

15. Issue a valid Set Idle request for the Report ID and duration being tested.

16. Issue a valid Get Idle request for the Report ID being tested.

17. If both requests complete successfully check that Get Idle request returns the correct value that was set.

18. The test is repeated for each possible device configuration. For HS capable device it is run with the device operating at both FS and HS.

Results Interpretation

The test transcribes all results to a text based log file.

<u>The test *fails* if:</u>

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Valid get HID descriptor commands fail for any reason.

Valid get HID report descriptor commands fail for any reason.

Set IDLE values do not match those returned by Get IDLE.

3.5.3 HID Get/Set Protocol Test

This test verifies that a boot class HID interface correctly supports the Get and Set Protocol HID Requests.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid Set Configuration command for the configuration to be tested (unless it is an other speed configuration descriptor).

2. Issue a valid Get Configuration command and verify that the configuration has been set. (if we are not testing an other speed configuration descriptor)

3. Issue a valid Get Configuration Descriptor with a requested length of 9 bytes.

4. Issue a valid Get Configuration Descriptor with a requested length of the wTotalLength field from the descriptor obtained in step 3.

5. For the configuration descriptor obtained in step 4 parse the descriptor to locate all interface descriptors.

6. For each interface descriptor check to see if bInterfaceClass == 0x03 (HID_CLASS)

7. For each interface descriptor that is HID check the bInterfaceSubClass. If the bInterfaceSubClass == 0x01 and bAlternateSetting == 0 perform the following test:

GET/SET PROTOCOL:

8. Issue a valid Set Protocol request for the Report Protocol.

9. Issue a valid Get Protocol request.

10. Check that the Get Protocol request returns Report Protocol.

11. Issue a valid Set Protocol Request for the Boot Protocol.

12. Issue a valid Get Protocol Request for the Boot Protocol.

13. Check that the Get Protocol request returns Boot Protocol.

14. The test is repeated for each possible device configuration. For HS capable device it is run with the device operating at both FS and HS.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Valid get HID descriptor commands fail for any reason.

Valid get HID report descriptor commands fail for any reason.

Set Protocol values do not match those returned by Get Protocol.

Valid Get or Set Protocol commands to a boot interface fail for any reason.

3.5.4 HID Report Descriptor Test

This test verifies that the device under test provides a Report Descriptor that meets the HID specification for each of its HID interfaces.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid Set Configuration command for the configuration to be tested (unless it is an other speed configuration descriptor).

2. Issue a valid Get Configuration command and verify that the configuration has been set. (if we are not testing an other speed configuration descriptor)

3. Issue a valid Get Configuration Descriptor with a requested length of 9 bytes.

4. Issue a valid Get Configuration Descriptor with a requested length of the wTotalLength field from the descriptor obtained in step 3.

5. For the configuration descriptor obtained in step 4 parse the descriptor to locate all interface descriptors.

6. For each interface descriptor check to see if bInterfaceClass == 0x03 (HID_CLASS)

7. For each interface found in Step 6 the following descriptor parsing is performed:

REPORT DESCRIPTOR PARSING

8. Issue a valid Get HID descriptor for the interface being tested.
9. Parse the HID descriptor to find the length of the report descriptor for this interface.
10. Issue a valid Get Report Descriptor request with a requested length from step 9.
11. Count the number of item tags in the Report Descriptor. It must be at least 1.

12. Parse the report descriptor according to section 6.2.2 of the HID specification.

13. The test is repeated for each possible device configuration. For HS capable device it is run with the device operating at both FS and HS.

Results Interpretation

The test transcribes all results to a text based log file.

<u>The test *fails* if:</u>

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

Valid get HID descriptor commands fail for any reason.

Valid get HID report descriptor commands fail for any reason.

A report descriptor does not comply with section 6.2.2 of the HID specification.

3.5.5 HID Specification Version Test

This test checks to see which version of the HID specification the device complies with.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Issue a valid Set Configuration command for the configuration to be tested (unless it is an other speed configuration descriptor).

2. Issue a valid Get Configuration command and verify that the configuration has been set. (if we are not testing an other speed configuration descriptor)

3. Issue a valid Get Configuration Descriptor with a requested length of 9 bytes.

4. Issue a valid Get Configuration Descriptor with a requested length of the wTotalLength field from the descriptor obtained in step 3.

5. For the configuration descriptor obtained in step 4 parse the descriptor to locate all interface descriptors.

6. For each interface descriptor check to see if bInterfaceClass == 0x03 (HID_CLASS)

7. For each interface found in Step 6 the following parsing is performed:
HID SPEC VERSION PARSING

8. Starting at the HID descriptor found in step 6 parse forward in the configuration descriptor to locate each of the following items:

    d. First HID descriptor after the HID interface.

    e. Second HID descriptor after the HID interface

    f. First EndPoint Descriptor after HID interface.

    g. First interface descriptor (any kind) after the HID interface.

9. If the location of A is greater than the location of D the test fails.
10. If the location of B is less than the location of D the test fails.
11. If the location of C is greater than the location of D the test fails.

12. If the location of A is less than C the interface is HID spec 4 compliant. Otherwise it is HID spec 3 compliant.

13. The test is repeated for each possible device configuration. For HS capable device it is run with the device operating at both FS and HS.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Device enumeration fails following the method described In this specification.

Valid get descriptor commands fail for any reason.

Valid set address commands fail for any reason.

Valid set configuration commands fail for any reason.

Valid get configuration commands fail for any reason.

A HID interface does not have at least one endpoint.

Two HID descriptors describe a single HID interface.

A HID interface does not have a following HID descriptor before the next interface.

3.6 USB On The Go (OTG) Tests

The following set of USB On The Go (OTG) Tests are run on OTG devices (dual role). The tests share a common algorithm to verify that the device under test is an OTG device. The algorithm is listed below. It is used by USBCV to determine whether the OTG tests should be run for the device under test.

OTG Class Algorithm:

1. Issue a valid Get Device Descriptor command to the device.
2. For each configuration that the device supports issue a valid get configuration descriptor call with a requested length of 9. Next issue a valid get configuration command with a request length of the wTotalLength field of the configuration descriptor.
3. For each configuration descriptor obtained in step 2 parse the descriptor finding all descriptors. If any of the descriptors are of OTG type (9) the device is an OTG device and the tests should be performed.

3.6.1 OTG Descriptor Test

This test verifies that the OTG device or SRP capable peripheral under test returns an OTG descriptor in response to any valid Get Configuration request. The user must indicate whether the device is an OTG device (dual role).

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid get configuration descriptor command with a requested length of 9 bytes.
3. Perform the following checks on the configuration descriptor value:
    bLength == 9
    bDescriptorType == CONFIGURATION descriptor type (or OTHER_SPEED_CONFIGURATION type)
4. Issue a valid get configuration descriptor command with a requested length of wTotalLength from the configuration descriptor
5. Parse the data returned and perform the following check on each descriptor:
    Check for type OTG descriptor (9)
6. The test fails if any of the following conditions are true:
    An OTG device is being tested and it has zero OTG descriptors.
    An OTG device is being tested and it has multiple OTG descriptors
    An SRP capable peripheral is being tested and it has zero OTG descriptors.
    An SRP capable peripheral is being tested and it has multiple OTG descriptors.
7. The following checks are performed on each OTG descriptor:
    bLength == 3
    Bits D2:D7 of bmAttributes must be zero.
8. For an SRP capable peripheral the SRP support field must be one and the HNP support field must be zero.
9. For an OTG device SRP support must be one and HNP support must be one.
10. If the test supports multiple configurations repeat test for each configuration descriptor.
11. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation and is also run for the OTHER speed configuration descriptor.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

Any valid Get Descriptor (Configuration) call fails.
An OTG descriptor does not have a bLength field equal to 3.
An OTG Device does not have an OTG Descriptor in any configuration.
An SRP capable device does not have an OTG Descriptor in any configuration.
An OTG Device or SRP capable device returns multiple OTG descriptors in response to a Get Descriptor (Configuration).

An OTG Device does not have HNP and SRP support set in the OTG descriptor.
A reserved bit in the OTG descriptor is not zero.
A SRP capable peripheral does not have SRP support set in an OTG descriptor.

3.6.2 Set Feature B_HNP_ENABLE

This test verifies that the OTG device under test correctly accept the Set Feature B_HNP_Enable command in the default and addressed states.

Device States For Test

This test is run with the device in the Default and Addressed states.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid Set Feature B_HNP_ENALBE command.
3. Check that the device successfully acknowledges the command.
4. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

<u>The test *fails* if:</u>

A valid Set Feature B_HNP_ENABLE command is not acknowledged in the Default or Address states.
An SRP only capable device does NOT stall the Set Feature B_HNP_ENABLE request

3.6.3 Set Feature A_HNP_SUPPORT

This test verifies that the OTG device under test correctly accept the Set Feature A_HNP_SUPPORT command in the default and addressed states.

Device States For Test

This test is run with the device in the Default and Addressed states.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid Set Feature A_HNP_SUPPORT command.
3. Check that the device successfully acknowledges the command.
4. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

<u>The test *fails* if:</u>

A valid Set Feature A_HNP_SUPPORT command is not acknowledged in the Default or Address states.

An SRP only capable device does NOT stall the Set Feature A_HNP_SUPPORT request

3.6.4 Set Feature A_ALT_HNP_SUPPORT

This test verifies that the OTG device under test correctly accept the Set Feature A_HNP_SUPPORT command in the default and address states.

Device States For Test

This test is run with the device in the Default and Addressed states.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid Set Feature A_ALT_HNP_SUPPORT command.
3. Check that the device successfully acknowledges the command.
4. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

A valid Set Feature A_ALT_HNP_SUPPORT command is not acknowledged in the Default or Address states.

An SRP only capable device does NOT stall the Set Feature A_ALT_HNP_SUPPORT request

3.6.5 Set Feature B_HNP_ENALBE and A_HNP_SUPPORT

This test verifies that the OTG device under test correctly accepts various combinations of the Set Feature B_HNP_ENABLE and Set Feature A_HNP_SUPPORT commands.

Device States For Test

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

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid Set Feature A_HNP_SUPPORT command.
3. Check that the device successfully acknowledges the command.
4. Issue a valid Set Feature B_HNP_ENABLE command.
5. Check that the device successfully acknowledges the command.
6. Repeat the test with the commands sent in the following states.

A_HNP_SUPPORT	Default	B_HNP_ENABLE	Default
A_HNP_SUPPORT	Default	B_HNP_ENABLE	Address
A_HNP_SUPPORT	Default	B_HNP_ENABLE	Configured
A_HNP_SUPPORT	Address	B_HNP_ENABLE	Address
A_HNP_SUPPORT	Address	B_HNP_ENABLE	Configured

7. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

A valid Set Feature A_HNP_SUPPORT command is not acknowledged in the Default or Address states.

A valid Set Feature B_HNP_ENABLE command is not acknowledged in the Default, Address, or Configured states.

3.6.6 Set Feature A_ALT_HNP_SUPPORT and A_HNP_SUPPORT

This test verifies that the OTG device under test correctly accepts various combinations of the Set Feature A_HNP_SUPPORT and Set Feature A_ALT_HNP_SUPPORT commands.

Device States For Test

This test is run with the device in the Default and Address states.

Overview of Test Steps

The test software performs the following steps.

1. Place the device in the desired starting state.
2. Issue a valid Set Feature A_ALT_HNP_SUPPORT command.
3. Check that the device successfully acknowledges the command.
4. Issue a valid Set Feature A_HNP_SUPPORT command.
5. Check that the device successfully acknowledges the command.
6. Repeat the test with the commands sent in the following states.

A_ALT_HNP_SUPPORT	Default	A_HNP_ENABLE	Default
A_HNP_SUPPORT	Default	A_ALT_HNP_SUPPORT	Default
A_HNP_SUPPORT	Default	A_ALT_HNP_SUPPORT	Address
A_ALT_HNP_SUPPORT	Default	A_HNP_SUPPORT	Address
A_ALT_HNP_SUPPORT	Address	A_HNP_ENABLE	Address
A_HNP_SUPPORT	Address	A_ALT_HNP_SUPPORT	Address

7. If the device is a high speed capable device the test must be run with the device in both high speed and full speed operation.

Results Interpretation

The test transcribes all results to a text based log file.

The test *fails* if:

A valid Set Feature A_HNP_SUPPORT command is not acknowledged in the Default or Address states.

A valid Set Feature A_ALT_HNP_SUPPORT command is not acknowledged in the Default, Address.

4. Change Log

4.1 Version 1.41

Updated to conform to USB 3.2 test spec:

- TD 9.3 is now required
- TD 9.28 and TD 9.29 are added (not required for USB 2 devices)
- TD 9.30 is added (required for all devices that must support BOS Descriptors (*bcdUSB* >= 0x200).

4.2 Version 1.42

Added assertions for Hub Global Test TD 2.19.