Universal Serial Bus
Device Class Definition
for
Device Bay Controllers

0.9rc5 Draft Revision
August 31, 1999

Scope of this Revision

The 0.8f revision reflects input from the founding members of the Class Working Group (see Contributors list below). The structure of this specification is stable, new members are joining the Class Working Group, and the content is being revised based on their feedback.

Contributors

John Dunn, Microsoft
Shaun Pierce, Microsoft
John Nels Fuller, Microsoft
Dan Shapiro, Microsoft
Jeff Stevens, Compaq
Charlie Shaver, Compaq
Chuck Stancil, Compaq
Paul Stanley, Compaq
Krunali Patel, Texas Instruments
George Soler, Microsoft
Paul Brant, SMSC
Steve Chang, KC Technology
Sue Vining, Texas Instruments
Jeff Enoch, Texas Instruments
Bryce Leach, Texas Instruments
Grant Ley, Texas Instruments
David Wooten, Compaq
Mark Williams, Microsoft
Clint Hanson, Granite Microsystems
E-mail: Paul.Brant@smsc.com

Revision History

Revision	Date; Filename	Author	Description
0.5	7/28/97	John Dunn	Base version
0.51a	8/2/97		Added usages for 1394, 1394/USB port mappings and power controls.
0.60	11/10/97	Dan Shapiro Shaun Pierce	Removed references to HID
0.61	12/18/97; Dbcclas1.doc	John Dunn, Shaun Pierce, Mark Williams	Added class-specific requests, edited descriptors, put into format expected by USB DWG.
0.62	12/19/97; Dbcclas2.doc	John Dunn, Mark Williams	Added “Notification Thru Interrupt Pipe” section.
0.62a	12/23/97; Dbcclas2a.doc	Jeff Stevens	Added DBC GUID, Bay x Port Mapping, and Power Capabilities descriptors
0.62b	1/6/98 ; Dbcclas2b.doc	Jeff Stevens Dan Shapiro Charlie Shaver	Added PHY reg’s and corrected power descriptors from Charlie’s comments. Merged w/ Dan Shapiro’s comments
0.62x	1/11/98; Dbccls2x.doc	John Dunn Mark Williams	Eliminated requests to write to DBC GUID register, Bay x Port Mapping Register, and Device Bay Power Capabilities Registers because these are read-only registers. Added requests the read and write the PHY control register. Moved definition of 1394 Config ROM content that DBC must support back into the Device Bay Spec.

			Moved Task Sequence Tables for USB-based DBC from Device Bay Spec to this spec and updated Sequence Table content with specific requests defined in this spec.
0.70	1/24/98	Jeff Stevens Charlie Shaver Chuck Stancil John Dunn Mark Williams	Made changes to make it clear DBC is self-powered USB function and not self-powered. Added Section 6 to include all requirements for USB-based DBC for one of its functions, which is to emulate a 1394 Link layer for its associated PHY. Added Section 7, informative appendix that shows a minimal Link controller implementation in a DBC. Added Section 8, informative appendix that shows an example implementation of a minimal Configuration ROM space in a DBC. Added vendor-specific requests mechanism. Reinstated Write CGUIDR and Write BPMRx requests; added constraint to Write DBCCR, Write CGUIDR, Write BPMRx, and Write DBPCRx requests. Walked through first draft of Task Sequence Tables and revised as necessary.
0.7a	2/3/98	John Dunn Mark Williams	Incorporated feedback from USB DWG breakout session in Atlanta; most of the changes are in section 5 and in the Task Sequence Table appendix.
0.8	2/24/98	John Dunn Mark Williams	Incorporated feedback from all e-mail received from attendees at USB DWG breakout session in Atlanta
0.8a	2/25/98	John Dunn George Soler Mark Williams	Rewrote “Management Overview” section to better describe relationship between the hub and the permanently attached DBC; moved static Form Factor fields from status bit-map to Bay Descriptor; redefined status bit-map fields; changed illustrations to show remote Device Bay application; reformatted Task Sequence Tables.
0.8b, c, and d	3/10/98	John Dunn Mark Williams George Soler	Miscellaneous edits based on feedback
0.8e	3/12/98	Mark Williams Jeff Stevens	A few edits based on feedback

0.9rc	3/18/98	Mark Williams	Edits requested by DBC Class Device Working Group at f2f meeting on Long Island, NY
0.9rc1	5/7/98	Mark Williams	Edits requested on last day of DBC Class Working Group at f2f meeting on Long Island, NY
0.9rc2	5/27/98	Mark Williams John Dunn	Added Get Bay Descriptor and Get Subsystem Descriptor class-specific requests in response to Sue Vining's e-mail; moved feature requests from section 5.4 to section 5.3; resolved "Note to Reviewers" undone items that were brought up at f2f meeting on Long Island, NY
0.8f	6/17/98	Mark Williams John Dunn Jeffrey Stevens Chuck Stancil Paul Brant	Bay subsystem wakup event changes. Removed USB Device wakup events section. Minor clarifications and typo's. Modified disclaimer information
0.9rc3	8/3/98	John Dunn Jeffrey Stevens Chuck Stancil Paul Brant Grant Ley	Consolidated diagrames. Added diagrams to clarify DBC USB Hub implementations. Added Examples.; Changed Subsystem Descriptor Power length fields; Removed requirement got 1394 phy to generate interrupt conditions;
0.9rc4	8/20/98	Paul Brant Chuck Stancil	Modify Subsystem Descriptor – Add programable Device Debounce time before setting DEVSTSCHG bit + transitioning to Device Inserted state. Define additional state in the Bay Status Bit-map - Device De-bounce.
0.9rc5	8/31/1999	Paul Brant John Dunn Grant Ley Chuck Stancil	Fixed subsystem desc. Offsets Added ENABLE_VOP_POWER value to the Set Feature request Added additional state transition "De-bounce" to the Section 9 Appendix Changed USB-Based DBC Insertion flow chart to table format

USB Device Class Definition for Device Bay Controllers

Please send comments via electronic mail to usbdevice@mailbag.intel.com

Table of Contents

1 Introduction

1.1 Scope

The Universal Serial Bus Device Class Definition for Device Bay Controllers (this specification) applies to all implementations of a Device Bay Controller (DBC) that communicate with the system using USB. Any DBC that appears as a USB device or sends its control signals across USB must comply with this specification in order to be Device Bay-compliant.

The Universal Serial Bus Device Class Definition for Device Bay Controllers does not apply to DBCs that interface to the system through the Advanced Control and Power Interface (ACPI). For more information about ACPI-based DBCs, see the Device Bay Specification.

In case of a conflict between this document and the Device Bay Specification, the Device Bay Specification shall have precedence.

1.2 Purpose

The purpose of this document is to describe the minimum capabilities and characteristics that a USB-based Device Bay Controller device must possess. This document also provides recommendations for optional features.

1.3 Related Documents

- Universal Serial Bus Specification, revision 1.1 (also referred to as the USB Specification). In particular, see Chapter 9, “USB Device Framework”. See www.usb.org/developers/
- Device Bay Interface Specification, Revision 0.85, or later.
- IEEE 1394 -1995 or later Terms and Abbreviations

This section defines terms used throughout this document. For additional terms that pertain to the Universal Serial Bus, see Chapter 2, “Terms and Abbreviations,” in the USB Specification and “Definitions of Terms” in section 1 of the Device Bay Interface Specification.

Device Bay subsystem
One or more Device Bay compliant bays controlled by a single Device Bay Controller.

Device Bay hub
An USB hub which provides the USB port connectivity for the USB Device Bay Controller and the bays

2 Management Overview

Every Device Bay subsystem requires one DBC.

A USB DBC must:

- Manage device insertion events.
- Manage device removal events.
- Associate inserted devices with downstream ports on its hub.
- Manage staged power consumption by the inserted devices.

This specification defines a DBC that is a USB function. The USB Core Specification defines a function as a device that is able to transmit or receive data or control information over the USB bus.

2.1 Device Bay Subsystem Architecture Containing a USB DBC Function

A DBC function uses USB to interface with the host.
The DBC and the bays it controls are all attached to ports on the same hub. Other ports on the hub can have other uses; for example, other ports on the hub can provide walk-up connectors for other USB devices, as shown in Figure 2.1.

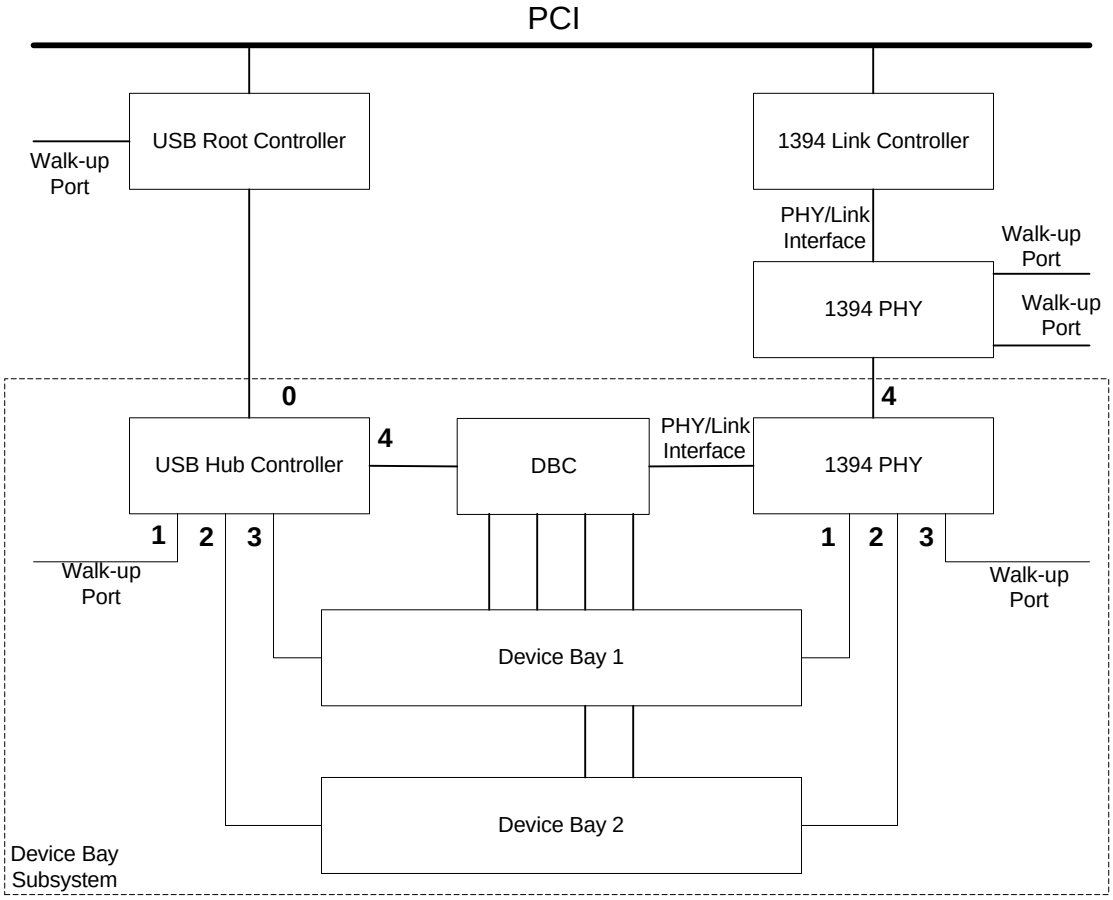

Figure 2-1. Remote Device Bay Architecture Showing Relationships Between USB Hub, DBC, 1394 PHY Link Interface and Bays

NOTE: Port numbers were arbitrarily assigned for use as with and example Bay Descriptor (See Descriptors chapter)

A DBC function is permanently attached to a port on a USB hub. A DBC function may be integrated in the same physical package as the hub, but is not required.
The Device Bay hub is the hub directly upstream from the Device Bay Controller and Device Bay USB ports.
Examples of a correct and incorrect implementation of the Device Bay hub are illustrated in Figure 2-2 and 2-3.

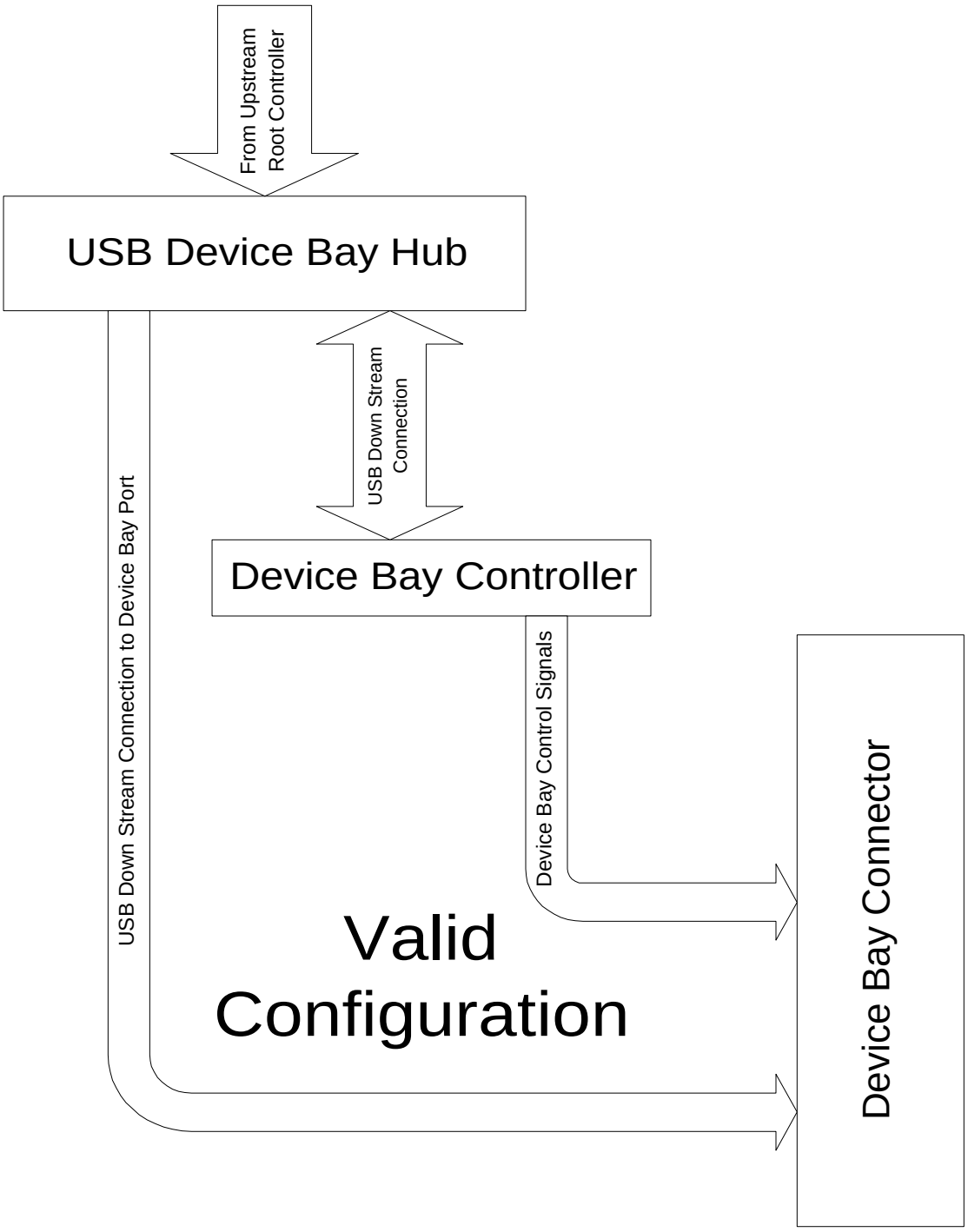

Figure 2-2. Example of a Valid Remote Device Bay Architecture Showing Relationships Between USB Hub, DBC, and Device Bay Connector

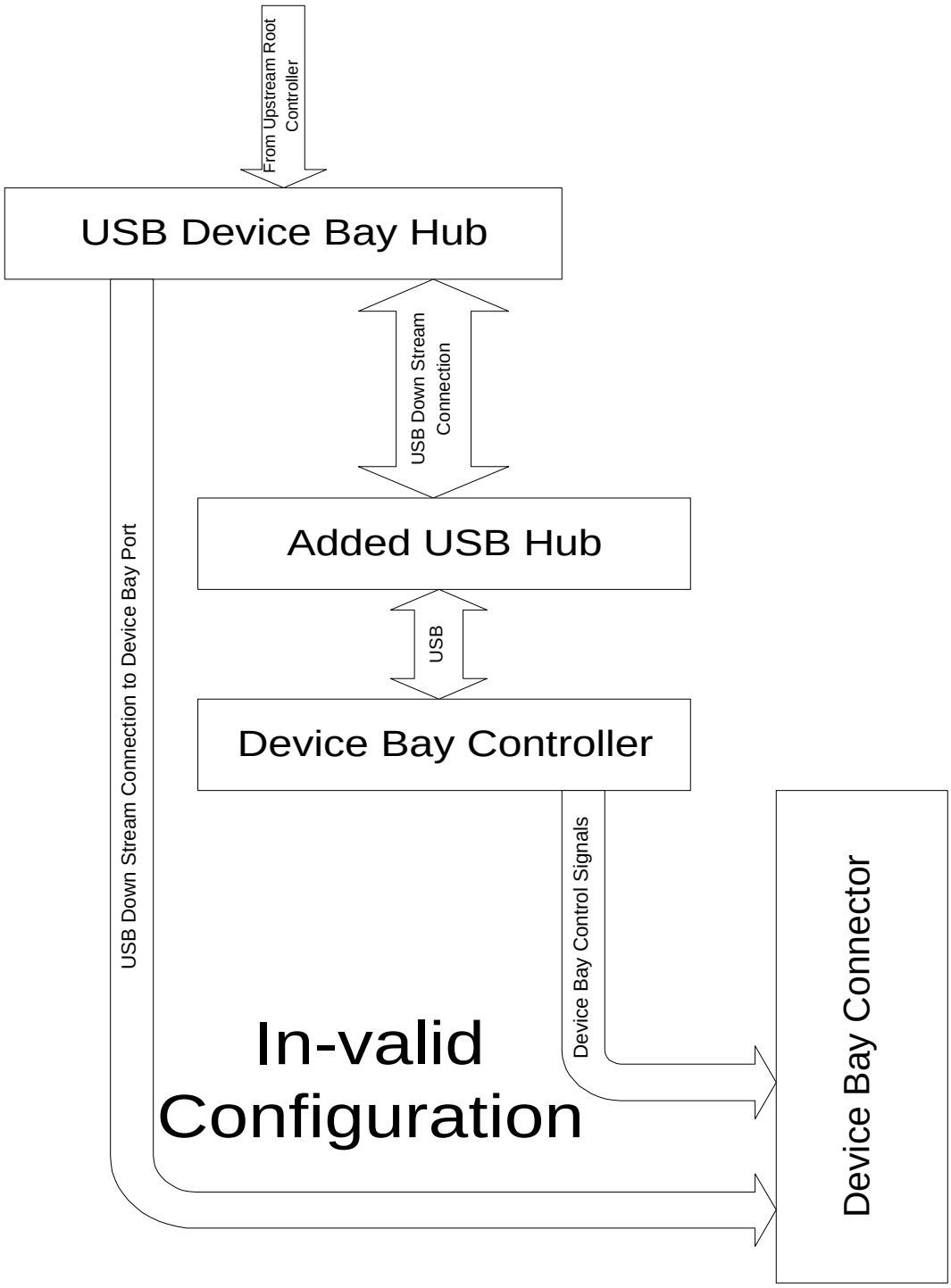

Figure 2-3. Example of an Invalid Remote Device Bay Architecture Showing Relationships Between USB Hub, DBC, and Device Bay Connector

2.2 Hub and DBC Power Distribution and Power Switching

This section specifies the rules for power distribution to the hub and permanently attached DBC function, as well as the rules for port power switching on the hub.

2.2.1 Power Distribution

The DBC function is permanently attached to a hub.

- The hub must be a "hybrid" powered device (as defined in sections 7.2.1.2 and 7.2.1.5 of the v1.1 USB Specification) or a self-powered device. A hybrid-powered hub has the advantage that communication from the host is possible even if the remote Device Bay power supply is off and is strongly recommended.
- The DBC function can be bus-powered or self-powered. Note that if the DBC is bus-powered, it must be able to operate with 500uA of current when the hub is suspended.

The DBC controls Vid to the device bay and may optionally control Vop.

2.2.2 Power Switching

It is recommended that the hub have port power switching for its downstream ports. An implementation may use individual port power switching or ganged power switching. For more information, see section 11.7 of the v1.1 USB Specification. Note that a hub port must be powered on in order to perform connect detection from the downstream direction.

- It is recommended that individual port power switching be used on a hub with ports that are attached to a DBC and one or more bays. The power-switching mode of a port is specified in the *wHubCharacteristics* field of the Hub Descriptor (for more information, see section 11.11.2.1 of the v1.1 USB Specification).
- If gang-switching is used, the *PortPwrCtrlMask* field of the Hub Descriptor must be used to mask all the ports attached to a DBC or bay from the effects of a gang-mode power control request.

Note that implementing either of these power switching modes on the hub enables one or more of the hub ports to provide a walk-up connector for a hot-plugged low-power or self-powered USB device.

Note to reviewers: a section will be added to the Device Bay Specification that defines a power descriptor for USB devices that are designed to be inserted into Device Bay bays.

2.2.3 Bay Subsystem Wakeup Events

For suspend, resume, and remote wakeup, the DBC is no different than any other USB device and must comply with the USB Core Specification requirements.

When enabled, the events at the bay subsystem that can be detected by the DBC and that cause the DBC to generate a USB wakeup event (that is, to drive resume signaling on its upstream USB port):

- When the DBC detects a device insertion event at the bay subsystem it controls, it must drive resume signaling on its upstream port.
- When the DBC detects a device removal request it must drive resume signaling on its upstream port when enabled via the REMOVE_REQUEST_ENABLE feature selector.
- When the DBC detects a device removal event at the bay subsystem it controls, it must drive resume signaling on its upstream port.

USB resume is enabled by the remote wakeup feature of the device status word. See chapter 9 of the USB 1.1 specification for additional information.

3 Functional Characteristics

Software running on the Host System to which the DBC is attached, uses the default control pipe to read and write Device Bay subsystem capabilities, status, and control information.

An interrupt pipe is required to enable the Device Bay subsystem to deliver information to the host about asynchronous, infrequent device insertion and removal request events (and, optionally, a vendor-specific notification).

3.1 Notification Through the Interrupt Pipe

Software running on the host is notified of an insertion and removal event when it receives a bit-map through the IN interrupt pipe. The only information contained in the event notification bit-map is an indication of the bay where the event occurred. One bit is set in the bit-map for each bay that has a pending interrupt:

If bit 0 is set, the event is a vendor specific event
If bit 1 is set, the event occurred at bay number 1
If bit 2 is set, the event occurred at bay number 2
If bit 3 is set, the event occurred at bay number 3
and so on.

Note: Please refer to section 5.2 for additional information about vendor specific events.

The DBC must assert the notification until the host acknowledges it. The host software acknowledges the notification by using the appropriate Clear Feature request.

- If host software determines the cause of the notification was a device status change, then the *bRequest* field of the Clear Feature must be set to C_DEVICE_STATUS_CHANGE.
- If host software determines the cause of the notification was the pressing of the removal request button, then the *bRequest* field of the Clear Feature request must be set to C_REMOVE_REQUEST

Host software gets the information it needs to determine the cause of the notification by using a Get Status request with the *bRequest* field set to BAY_STATUS. The *wIndex* field of the Get Status request is set to the number of the bay that was indicated in the bit-map on the Interrupt pipe. Bays in a Device Bay subsystem are numbered from 1 to n.

3.2 1394 Link Requirements

A USB-based DBC must have a 1394 Link/PHY interface and simple Link controller functionality (the relationship between the USB-based DBC and its associated PHY is shown in the architecture diagram shown in Figure 2-1. More specifically, a USB-based DBC must

- Provide Link layer services; that is, handle the request, indication, response, and confirmation service primitives described in section 3.6.1 of the IEEE Std 1394-1995 specification.
- Provide 1394 CSR space and Configuration ROM.

For more information about the minimal requirements see section 6 of this specification.

4 Descriptors

This section describes the standard and class-specific USB descriptors for the Device Bay Controller class.

4.1 Device Descriptor

4.1.1 Class-Specific Device Descriptor

The class-specific device descriptors for a DBC Class device are:

- Subsystem Descriptor
- Bay Descriptor

The Subsystem Descriptor and Bay Descriptor must be returned after the Interface Descriptor and before the Endpoint Descriptors for the DBC device.

Example:

1. Config_Descriptor
   Interface_descriptor
   Subsystem_descriptor (class)
   Bay1_descriptor
   Bay2_descriptor
   …
   Bayn_descriptor
   Endpoint_descriptor

4.1.1.1 Subsystem Descriptor

The Subsystem Descriptor contains:

- The number of bays in the subsystem.
- Whether or not any bays in the subsystem implement a physical security lock.
- The unique identifier of the 1394 PHY for which the DBC is providing 1394 Link services.
- The power capabilities of the subsystem.
- The USB DBC Class specification version to which the subsystem complies.
- The Device Debounce state timeout value
- Whether VOP switching support is provided for one or more bays

Table 4-1. Subsystem Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor, in bytes (must be at least 0x30).
1	*bDescriptorType*	1	Constant	SUBSYSTEM descriptor type (must be 0x40).
2	*bmAttributes*	4	Bit Map	Attributes of the Device Bay Subsystem controlled by the DBC. D 31…D12: Reserved, must be 0 D 11…D8: The amount of time that the DBC waits in the Device Debounce state before setting the DEVSTSCHG bit and transitioning to the Device Inserted state. The value in this field represents the de-bounce time in 0.5 second increments. A value of 0000b represents a debounce time of 0.5 seconds. A value of 1111b represents a debounce time of 8.0 seconds. D5: VOP switching support. 0 = No VOP switching support is

				provided in this subsystem. 1 = VOP switching support is provided in this subsystem. D 4: Physical security lock support. 0 = No physical security locks in this subsystem 1 = At least one bay in the susbsystem has a physical security lock. D 3..0: The number of bays in the Subsystem.
6	*dw1394LinkGUID*	8	Number	Contains the 64-bit GUID (in big endian format) of the 1394 PHY for which the DBC is providing Link services.
14	*dw3_3ContinuousPower*	4	Number	Total amount of continuous power, at 3.3V, available to the susbsystem. Measured in milliwatts. For a definition of the continuous power measurement, see the Device Bay Specification.
18	*dw3_3PeakPower*	4	Number	Total amount of peak power, at 3.3V, available to the subsystem. Measured in milliwatts. For a definition of the peak power measurement, see the Device Bay Specification.
22	*dw5_0ContinuousPower*	4	Number	Total amount of continuous power, at 5.0V, available to the subsystem. Measured in milliwatts.
26	*dw5_0PeakPower*	4	Number	Total amount of peak power, at 5.0V, available to the subsystem. Measured in milliwatts.
30	*dw12_0ContinuousPower*	4	Number	Total amount of continuous power, at 12.0V, available to the subsystem. Measured in milliwatts.
34	*dw12_0PeakPower*	4	Number	Total amount of peak power, at 12.0V, available to the subsystem. Measured in milliwatts.
38	*dwaggregatePower*	4	Number	The total aggregate power available to the subsystem. Measured in Watts. For a definition of total aggregate power, see the Device Bay specification.
42	*dwthermalDissipation*	4	Number	The total amount of heat that can be removed from the subsystem. Measured in Watts.
46	*bcdSpecificationRelease*	2	BCD	The USB DBC Class specification release to which the subsystem complies (assigned by vendor).

4.1.1.2 Bay Descriptor

There is one Bay Descriptor for each bay in the subsystem.

A Bay Descriptor contains

- A unique identifier for the bay within the subsystem (1, 2, 3, and so on). See illustration below.
- The 1394 port to which the bay is connected (see illustration below).
- The USB hub port to which the bay is connected (see illustration below).
- The form factor of the bay

Table 4-2. Bay Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor, in bytes (must be 0x6).
1	*bDescriptorType*	1	Constant	BAY descriptor type (must be 0x41).
2	*bBayNumber*	1	Byte	The unique identifier for the bay within the susbsystem (1, 2, 3, and so on).

3	*bHubPortNumber*	1	Byte	Identifies the USB hub port to which the bay is connected.
4	*bPHYPortNumber*	1	Byte	Identifies the PHY port to which the bay is connected.
5	*bFormfactor*	1	Bit Map	Form factor of the bay: 0x00=DB32 0x01=DB20 0x02=DB13 All other values reserved.

Figure 2-1 provides an illustration of an example bay subsystem configuration and how the Bay Descriptor fields are used. The port assignments in the example arbitrarily chosen for use in the Bay Descriptors example below. For example, referring to Figure 2-1, if Bay 1 is a DB32 form factor, then the Bay 1 Descriptor values would be as follows:

Offset	Field	Value
0	*bLength*	0x06
1	*bDescriptorType*	0x41
2	*bBayNumber*	0x01
3	*bHubPortNumber*	0x03
4	*bPHYPortNumber*	0x01
5	*bFormfactor*	0x00

4.1.2 Standard Device Descriptor

The standard device descriptor for a DBC Class device must indicate that class information is to be found at the interface level. Therefore, the *bDeviceClass* field of the standard device descriptor for a DBC Class device must contain zero so that enumeration software looks down at the interface level to determine the Interface Class.

The *bDeviceSubClass* and *bDeviceProtocol* fields for a DBC Class device descriptor must be set to zero.

All other fields of the standard device descriptor must comply with the definitions in section 9.6.1 of the USB Specification.

Table 4-3. Standard Device Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor, in bytes (must be 0x12).
1	*bDescriptorType*	1	Constant	DEVICE descriptor type (must be 0x01).
2	*bcdUSB*	2	BCD	Identifies the version of the USB Specification that the DBC and its descriptors are compliant with.
4	*bDeviceClass*	1	Class	Must be 0x00 for a DBC Class device.
5	*bDeviceSubclass*	1	Subclass	Must be 0x00 for a DBC Class device.
6	*bDeviceProtocol*	1	Protocol	Must be 0x00 for a DBC Class device.
7	*bMaxPacketSize0*	1	Number	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid).
8	*idVendor*	2	Number	Vendor ID (assigned by USB).
10	*idProduct*	2	Number	Product ID (assigned by vendor).
12	*bcdDevice*	2	BCD	Device release number (assigned by vendor).
14	*iManufacturer*	1	Index	Index of string descriptor describing manufacturer.
15	*iProduct*	1	Index	Index of string descriptor describing product.
16	*iSerialNumber*	1	Index	Index of string describing the device's serial number.
17	*bNumConfigurations*	1	Number	Number of possible configurations (must be 0x01 for DBC Class device).

4.2 Configuration Descriptor

4.2.1 Class-Specific Configuration Descriptor

There is no class-specific configuration descriptor.

4.2.2 Standard Configuration Descriptor

A DBC Class device Configuration Descriptor is identical to the standard configuration descriptor defined in section 9.6.2 of the USB Specification.

Table 4-4. Standard Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor, in bytes (must be 0x09).
1	*bDescriptorType*	1	Constant	CONFIGURATION descriptor type (must be 0x02).
2	*wTotalLength*	2	Length	The combined length of all descriptors (configuration, interface, endpoint, and class or vendor specific) returned for this configuration.
4	*bNumInterfaces*	1	Number	Number of interfaces supported by this configuration (must be 0x01 for DBC Class device).
5	*bConfigurationValue*	1	Number	Value to use as an argument for SetConfiguration to select this configuration.
6	*iConfiguration*	1	Index	Index of string descriptor describing this configuration.
7	*bmAttributes*	1	Bit Map	D7: Bus Powered D6: Self Powered D5: Supports Wakeup D4 – D0: Reserved For a DBC Class device, can be set to 0x60 (self-powered and supports wakeup) or 0xA0 (bus-powered and supports wake-up).
8	*maxPower*	1	mA	Maximum power consumption from the bus when DBC is fully operational (expressed in 2mA units). This value cannot exceed 50 (100mA).

4.3 Interface Descriptor

4.3.1 Class-Specific Interface Descriptor

There is no class-specific interface descriptor.

4.3.2 Standard Interface Descriptor

A DBC Class device Interface Descriptor is identical to the standard interface descriptor defined in section 9.6.3 of the USB Specification.

Table 4-5. Standard Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor, in bytes (must be 0x09).

1	*bDescriptorType*	1	Constant	INTERFACE descriptor type (must be 0x04).
2	*bInterfaceNumber*	1	Number	Zero-based value that identifies the index in the array of concurrent interfaces supported by this configuration (must always be 0x00 for DBC Class devices).
3	*bAlternateSettings*	1	Number	Value used to select alternate settings (must always be 0x00 for DBC Class devices).
4	*bNumEndPoints*	1	Number	Number of endpoints used by this interface (must always be 0x01 for DBC Class devices).
5	*bInterfaceClass*	1	Class	Interface class code (0xFF or TBD value assigned by USB).
6	*bInterfaceSubClass*	1	SubClass	Subclass code (must always be 0x00 for DBC Class devices).
7	*bInterfaceProtocol*	1	Protocol	Protocol code (must always be 0x00 for DBC Class devices).
8	*iInterface*	1	Index	Index of string descriptor describing this interface.

4.4 Endpoint Descriptor

4.4.1 Class-Specific Endpoint Descriptor

There is no class-specific endpoint descriptor.

4.4.2 Standard Endpoint Descriptors

4.4.2.1 Control Endpoint Descriptor

Since endpoint 0 is used as the DBC control endpoint, there is no dedicated standard control endpoint descriptor.

4.4.2.2 Interrupt Endpoint Descriptor

A DBC Class device must have an interrupt endpoint.
The descriptor for this endpoint is identical to the standard endpoint descriptor defined in section 9.6.4 of the USB Specification.

Table 4-6. Interrupt Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor, in bytes (must be 0x07).
1	*bDescriptorType*	1	Constant	ENDPOINT descriptor type (must be 0x05).
2	*bEndpointAddress*	1	Endpoint	D7: Direction (1=IN) D6 – D4: Reserved D3 – D0: Endpoint number Must be 0x81 for a DBC Class device (an IN endpoint with an endpoint number of 1).
3	*bmAttributes*	1	Bit Map	D7 – D2: Reserved D1 – D0: Transfer type (Interrupt = 11b) Must be 0x03 for a DBC Class device.
4	*wmaxPacketSize*	2	Number	Maximum packet size this endpoint is capable of sending. For a DBC Class device, this value depends on the number of bays controlled by the device: if 1 to 7 bays then 0x0001, if 8 to 15 bays then 0x0002, if 16 to 23 bays then 0x0003, and so on.
6	*binterval*	1	Number	Interval for polling endpoint for data transfers (expressed in milliseconds). For a DBC Class device, the value of this field must be 32 (0x20).

5 Requests

This section specifies the requests that the host can send to the DBC. The requests are listed in the following summary table:

Table 5-1. USB DBC Requests

Brequest	Value	Description
GET_STATUS	0	See Chapter 9 of the USB Specification and section 5.3.1 of this specification.
CLEAR_FEATURE	1	See Chapter 9 of the USB Specification and sections 5.3.4.2 and 5.4.3 of this specification.
Reserved for future use	2	
SET_FEATURE	3	See Chapter 9 of the USB Specification and sections 5.3.4.1 and 5.4.3 of this specification.
Reserved for future use	4	
SET_ADDRESS	5	See Chapter 9 of the USB Specification.
GET_DESCRIPTOR	6	See Chapter 9 of the USB Specification.
SET_DESCRIPTOR	7	See Chapter 9 of the USB Specification.
GET_CONFIGURATION	8	See Chapter 9 of the USB Specification
SET_CONFIGURATION	9	See Chapter 9 of the USB Specification.
GET_INTERFACE	10	See Chapter 9 of the USB Specification.
SET_INTERFACE	11	See Chapter 9 of the USB Specification.
SYNCH_FRAME	12	See Chapter 9 of the USB Specification.
GET_PHY_REG	13	See section 5.3.2 of this specification.
SET_PHY_REG	14	See section 5.3.3 of this specification.

5.1 Standard Requests

The DBC Device Class supports the standard requests described in Chapter 9, “USB Device Framework,” of the USB Specification, as shown in Table 5-1.

5.2 Vendor-Specific Requests

Examples of vendor-specific requests relevant to Device Bay subsystems are:

- Over-current reporting.
- Over-temperature reporting.
- Asset tracking.
- Security measures.

Vendor-specific requests may be implemented in at least two different ways:

- The interrupt pipe required of all USB-based DBC implementations must be used to notify the host software of a vendor-specific request. Bit 0 of the bit-map sent from the device to the host is reserved for this: if bit 0 is set to 1, the host software will recognize this as a vendor-specific request.
- A second interrupt pipe may be added to the USB-based DBC that is dedicated to vendor-specific requests, which are handled as specified in the Chapter 9 of the USB Core Specification.

For more information, see section 3.1, “Notification through the Interrupt Pipe.”

5.3 Class-Specific Requests

This section specifies the class-specific requests for a DBC. If the DBC device gets an invalid request from the host, the device behavior is undefined; an invalid request can be handled by the device in whatever way is convenient for the developer. For example, it is not required to STALL the control pipe when an invalid request is received; the device can simply ignore the invalid request.

5.3.1 Get Bay Status

The Get Bay Status request returns the following information about a particular bay:

-
- Current state of the optional bay-mounted security lock.
- Current state of the bay (empty, device debounce, device enabled, device inserted, removal request pending, or device removal allowed).
- Current state of the removal request button mounted on the bay.
- Whether or not a USB device is currently inserted in the bay.
- Whether or not a 1394 device is currently inserted in the bay.
- Whether or not the status of a device in the bay has changed.
- Whether or not the software controlled interlock is currently engaged.
- The state of the bay, as requested by the host.
- Whether or not the hardware device removal request interrupt is currently enabled.
- Whether or not the hardware device change event interrupt is currently enabled.
- Whether or not the hardware device removal event interrupt is currently enabled.

BmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_STATUS	0	Index of bay (1, 2, 3, and so on)	3	Bit-map that indicates the current state of the bay

The format of the bit-map returned by the DBC in response to a Get Status request from the host with *wValue* set to BAY_STATUS is shown in the following table.

Table 5-1. Bay Status Bit-map

Bits	Description	Comment
23-16	Reserved	
15	Current state of the optional bay-mounted physical security lock: 0 = Physical security lock is either not implemented on this bay or it is Implemented and currently disengaged. 1 = Physical security lock is engaged.	The host can only read this bit.
14-12	Current state of the bay: 000 = Bay Empty 001 = Device Inserted 010 = Device Enabled 011 = Removal Requested 100 = Device Removal Allowed 101 = Device De-bounce All other values are reserved.	For more information about these device states, see section 6.6.6 of the Device Bay Specification. The host can only read this bit.
11	Current state of the optional removal request button on the bay: 0 = Removal request button is either not implemented on this bay or it is implemented and currently cleared (any presses of the button have been acknowledged by the host). 1 = Removal request button has been pressed and not yet acknowledged by the host.	The host can read this bit and clear this bit.
10	Current state of transitions on the device presence pins in the bay. 0 = Currently cleared (any transitions on either presence pin have been acknowledged by the host. 1 = A transition has occurred on one or both of the presence pins and has not yet been acknowledged by the host.	The host can read this bit and clear this bit. For more information about the device presence pins, 1394PRSN# and USBPRSN#, see section 4.4.1.4 of the Device Bay Specification.
9	Current state of the 1394 device presence pin in the bay: 0 = No 1394 device currently inserted in the bay. 1 = A 1394 device is currently inserted in	The host can only read this bit.

	the bay.	
8	Current state of the USB device presence pin in the bay: 0 = No USB device currently inserted in the bay. 1 = A USB device is currently inserted in the bay.	The host can only read this bit.
7	Current state of the software-controlled interlock:. 0 = Disengaged 1 = Engaged	The host can use Set Feature and Clear Feature requests to engage and disengage the interlock. For more information, see section 5.3.4.
6 - 4	Most recent bay state change request the DBC has received from the host: 000 = No op 001 = Request was for Device Inserted state 010 = Request was for Device Enabled state 011 = Request was for Removal Requested 100 = Request was for Device Removal Allowed	
3	Current state of the removal request event interrupt enable: 0 = Disabled 1 = Enabled	The host can use Set Feature and Clear Feature requests to enable and disable this interrupt. For more information, see section 5.3.4.
2	Current state of the device status change event interrupt enable: 0 = Disabled 1 = Enabled	The host can use Set Feature and Clear Feature requests to enable and disable this interrupt. For more information, see section 5.3.4.
1	Current state of the device removal wakeup interrupt enable: 0 = Disabled 1 = Enabled	The host can use Set Feature and Clear Feature requests to enable and disable this interrupt. For more information, see section 5.3.4.
0	Current state of the Vid control bit: 0 = Turn off Vid power. 1 = Turn on Vid power.	The host can use Set Feature and Clear Feature requests to enable and disable Vid power. For more information, see section 5.3.4.

5.3.2 Get PHY Register

Host software uses this request to establish a consistent communication method between the Link and PHY.

The *bmRequestType* field value specifies a Class-type request directed to an interface, with a data transfer direction of device to host.

The *wIndex* field is set to indicate the PHY register to read from.

The *wLength* field is always set to 1 because data is always read one byte at a time from a PHY register.

BmRequestType	**bRequest**	**wValue**	**wIndex**	**wLength**	**Data**
10100001B	GET_PHY_REG	0	The 4-bit address of the PHY register to read from.	1	Contains the byte of data read from the PHY register.

5.3.3 Set PHY Register

The host uses this request, along with the Get PHY Register request,

The *bmRequestType* field value specifies a Class-type request directed to an interface, with a data transfer direction of host to device.

The *wIndex* field is set to indicate the PHY register to write to.

The *wLength* field is always set to 1, because a byte of data is always written to a PHY register.

BmRequestType	BRequest	wValue	wIndex	wLength	Data
00100001B	SET_PHY_REG	0	The 4-bit address of the PHY register to write to.	1	Contains the byte of data to write to the PHY register.

5.3.4 Feature Requests

The Get Bay Status request (specified in section 5.3.1) enables the host to read all the information it needs from the DBC. Set Feature and Clear Feature requests enable the host to write information to the DBC.

5.3.4.1 Set Feature

The Set Feature requests available to host software are specified in this section. For a list of the feature selectors (*wValue* field values) that can be used in a Set Feature request, see section 5.3.4.3.

BmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_FEATURE	Feature Selector	Index of bay (1, 2, 3, and so on)	Zero	None

5.3.4.2 Clear Feature

The Clear Feature requests available to host software are specified in this section. For a list of the feature selectors (*wValue* field values) that can be used in a Clear Feature request, see section 5.3.4.3.

BmRequestType	BRequest	wValue	wIndex	Wlength	Data
00100001B	CLEAR_FEATURE	Feature Selector	Index of bay (1, 2, 3, and so on)	Zero	None

5.3.4.3 Feature Selector Values

The valid values for the *wValue* field in a Set Feature or Clear Feature request are listed in the following table.

Table 5-2. *wValue* Values for Set Feature

Feature Selector	Recipient	Value	Set/Clear	Description
DEVICE_STATUS_CHANGE_ENABLE	Interface	0	Set/Clear	On the specified bay, enables a notification due to a device status change event (for more information, see section 6.6.5 of the Device Bay Specification).

ENABLE_VID_POWER	Interface	1	Set/Clear	On the specified bay, enables the Vid rail for the bay (for more information, see section 6.6.5 of the Device Bay Specification).
LOCK_CTL	Interface	2	Set/Clear	On the specified bay, engages the software controlled interlock mechanism for the bay (for more information, see section 6.6.5 of the Device Bay Specification).
REMOVAL_EVENT_WAKE_ENABLE	Interface	3	Set/Clear	On the specified bay, enables a notification due to a removal event in the bay (for more information, see section 6.6.5 of the Device Bay Specification).
REMOVE_REQUEST_ENABLE	Interface	4	Set/Clear	On the specified bay, enables a notification due to a hardware removal request. For more information, see section 6.6.5 of the Device Bay specification.
REQUEST_DEVICE_INSERTED_STATE	Interface	5	Set	Request to change bay state to device inserted.
REQUEST_DEVICE_ENABLED_STATE	Interface	6	Set	Request to change the bay state to device enabled.
REQUEST_REMOVAL_REQUESTED_STATE	Interface	7	Set	Request to change the bay state to removal requested.
REQUEST_REMOVAL_ALLOWED_STATE	Interface	8	Set	Request to change the bay state to removal allowed.

C_DEVICE_STATUS_CHANGE	Interface	9	Clear	On the specified bay, acknowledges the notification that indicates the device status has changed (for more information, see section 6.6.4 of the Device Bay Specification).
C_REMOVE_REQUEST	Interface	10	Clear	On the specified bay, acknowledges the notification that indicates that the removal request button has been pressed (for more information, see section 6.6.4 of the Device Bay Specification).
ENABLE_VOP_POWER	Interface	11	Set/Clear	On the specified bay, enables the Vop rail for the bay (for more information, see section 6.6.5 of the Device Bay Specification).

6 Appendix (Normative)

This normative appendix describes the minimal DBC Link controller, CSR space, and Configuration ROM requirements. A USB-based DBC must

- Provide Link layer services; that is, handle the request, indication, response, and confirmation service primitives described in section 3.6.1 of the IEEE Std 1394-1995 specification.
- Provide 1394 CSR space and Configuration ROM.

6.1 Reporting PHY Interrupt Conditions

A minimal DBC is not required to report phy interrupt conditions.

6.2 Providing Link Layer Services

A USB-based DBC must handle the request, indication, response, and confirmation service primitives described in section 3.6.1 of the IEEE Std 1394-1995 specification. Section 7 of this specification shows one way to implement the minimal set of services that need to be provided by a USB-based DBC.

6.3 Providing 1394 CSR Space and Configuration ROM

The DBC must implement a minimal amount of control status register (CSR) space and Configuration ROM space to be a 1394 transaction-capable node. This section specifies the CSR space and Configuration ROM space that must be implemented.

6.3.1 CSR Space

The CSR core registers that must be implemented in the DBC for the DBC to function as a transaction-capable node are listed in Table 6-1. The base address of the register space is FFFF F000 0000_{16}. The registers are listed by the byte offset from the base address.

Table 6-1. CSR Core Registers

Offset	Register name	Description
0000_{16}	STATE_CLEAR	Sets state and control information. For more information, see section 8.3.2.2.1 of the IEEE Std 1394-1995 specification. • The *unit_depend* field is not required for the DBC. • The *lost* bit must be implemented in the DBC. In a cable environment, the *lost* bit is not affected by a bus reset, but is set to "1" during bus reset if a power reset or transition to the dead state occurs (as defined in the CSR Architecture). • The *dreq* bit must be implemented in the DBC. The DBC is a slave device on the 1394 bus and will not originate 1394 transactions, which means the *dreq* bit must be set to "1." The *dreq* bit is unaffected by a bus reset. • In the *bus_depend* field the DBC: The only bit in the *bus_depend* field implemented in the DBC is the *gone* bit. The DBC is not cable-powered so does not implement the *linkoff* bit. The *linkoff* bit is set to "0" in the DBC and must remain at zero. The DBC is not cycle-master capable and does not implement the *cmstr* bit. The *cmstr* bit is set to "0" in the DBC and must remain at zero.
0004_{16}	STATE_SET	Sets STATE_CLEAR bits. For more information, see section 8.3.2.2.2 of the IEEE Std 1394-1995 specification.
0008_{16}	NODE_IDS	Specifies 16-bit node ID. For more information, see section 8.3.2.2.3 of the IEEE Std 1394-1995 specification. • The DBC must not be used in a backplane environment. The behavior of the DBC in a backplane environment will not be specified.
$000C_{16}$	RESET_START	Resets state for a node. For more information, see section 8.3.2.2.4 of the IEEE Std 1394-1995 specification.

Note that

- The SPLIT_TIMEOUT register is not implemented in DBC Link controller implementations in which the Link controller is not a requester on the 1394 bus. The definition of the SPLIT_TIMEOUT register in the ANSI/IEEE Std 1212:1994 or ISO/IEC 13213:1994 specification states that only a requester must implement the SPLIT_TIMEOUT register if a device is implementing split transactions. If a DBC Link controller implements requester capability and split transaction capability, then this DBC Link controller implementation must implement the SPLIT_TIMEOUT register in accordance with the ANSI/IEEE Std 1212:1994 or ISO/IEC 13213:1994 specification.
- The BUSY_TIMEOUT register is not implemented in a minimal DBC Link controller; otherwise, it is optional. Because the BUSY_TIMEOUT register is not implemented, the DBC will not support retries if using a minimal Link controller implementation.

6.3.2 Providing Configuration ROM

The DBC must implement a Configuration ROM using the general ROM format. This section describes the parts of the general ROM format that the DBC implements.

The general ROM format used by the DBC is shown in the following figure:

info_length	crc_length	rom_crc_value

(8)	(8)	(16)
bus_info_block		
root_directory		
unit_directories		
root & unit leaves		
vendor_dependent_information		

The Configuration ROM must contain a bus_info_block, a root directory, and a unit directory.

Note that:

- Currently, no vendor_dependent_information is defined for the Configuration ROM of the DBC.
- The “root & unit leaves” information must consist of the Node_Unique_Id leaf required by the root directory structure. The unit_directory must contain entries that identify the DBC to the software.
- The *info_length* field must have a value greater than 1 and must specify how many quadlets are contained in the bus_info_block data structure.
- The *crc_length* field specifies how many quadlets are protected by the *rom_crc_value*. The minimum number for *crc_length* is the size of the bus_info_block, while the maximum value for *crc_length* is 255, which allows for a maximum Configuration ROM size of 1024 bytes.
- The *rom_crc_value* is calculated using the CRC-16 algorithm described in clause 8.1.5 of the CSR Architecture.

For more information about the Bus_Info_Block, the Root_Directory, and the Unit_Directory, see the IEEE Std 1394-1995 specification. Notes about these components of Configuration ROM space that are specific to the minimal DBC Link controller are listed in Table 6-2.

Table 6-2. Notes on DBC Link Controller Configuration ROM Space Components

Configuration ROM Component	Notes for DBC Link Controller
Bus_Info_Block	For a full description of the Bus_Info_Block, see the CSR Architecture specification. In the DBC Configuration ROM space: • The *cmc* bit must be zero. • The *isc* bit must be zero. • The *bmc* bit must be zero. • The *cyc_clk_acc* field must be set to all ones. • The *max_rec* field of the DBC must be set to 0001 (4 bytes); this is the only value allowed in the DBC for the *max_rec* field.
Root_Directory	For a full description of the Root_Directory, see the CSR Architecture specification. The Configuration ROM for the DBC must implement the general ROM format and must have a Root_Directory. The Root_Directory must contain the following entries: Module_Vendor_Id, Node_Capabilities, Node_Unique_Id, and Unit_Directory. The DBC shall implement the *64*, *fix*, *lst*, *spt*, and *drq* bits. • The *64*, *fix*, and *lst* bits must be set to 1 in the DBC. These bits indicate that the DBC uses a 64-bit fixed-addressing scheme and that the STATE_CLEAR.*lost* bit is implemented. • The DBC sets the *spt* bit to "0" to indicate that the DBC does not implement the SPLIT_TIMEOUT register. • The DBC must not initiate 1394 transactions and must set the *drq* bit to zero.
Unit_Directory	For a full description of the Root_Directory, see the CSR Architecture specification. The Unit_Directory must contain the minimum amount of information necessary to identify the DBC to the software. The Unit_Spec_ID entry and Unit_SW_Version entry must be in the Unit_Directory. • The value to use for the DBC *Unit _Spec ID* is 0x00805F. • The 24-bit value for *unit sw version* is 0x010000.

An example implementation of a DBC Configuration ROM is shown in Section 8.

7 Appendix (Informative)

This section details the minimal link controller behavior required for the DBC. Section 7.8 lists additional features that may be added to a DBC link controller implementation for a design that must go beyond the minimal requirements.

7.1 Minimal Link Controller Transaction Capability

The minimal USB-based DBC Link controller must be a transaction-capable 1394 node.

- The Link must participate in asynchronous transactions
- The Link does not need to recognize isochronous transfers.

Section 8.3.1.2 of the IEEE Std 1394-1995 specification lists the requirements for a transaction-capable node.

- The minimal Link controller is not required to implement the SPLIT_TIMEOUT register that is called out in section 8.3.1.2 of the IEEE Std 1394-1995 specification.
- The minimal Link controller is not required to support split transactions. If split transactions are not supported the minimal Link controller must be able to respond to read requests using concatenated subactions and must respond to write requests using a unified response.

In addition to the registers listed in the IEEE Std 1394-1995 specification, the Link controller also must have a Configuration ROM in the general ROM format (for more information, see section 6 of this specification).

7.2 1394 Packets

The PHY attached to the DBC Link controller must pass 1394 packets to the minimal Link controller in the DBC. A minimal Link controller must recognize only two types of packets from the 1394 bus and must generate only one type of packet to the 1394 bus.

7.2.1 Receiving Packets

The DBC Link controller must ignore a packet that contains a CRC error in the *header_CRC*. The *header_CRC* is the only CRC that the DBC Link controller must check, since the minimal Link controller will never respond to a packet with a data payload. The *max_rec* field of aDBC implementing a minimal link should be set to 0h, meaning the maximum payloadfor an asynchronous block transaction is not specified since block transactions are not supported. A minimal DBC Link controller must properly recognize and process packets with the transaction codes listed in Table 7-1.

Table 7-1. Packet Transaction Codes

Packet	Transaction code
Write request for data quadlet	0h
Read request for data quadlet	4h

In addition to the two packet types, a minimal DBC link controller must also properly handle broadcast packets. If the 10-bit bus_ID is between 000 and 3FEh, and the physical_ID is set to 3Fh, then this is a broadcast to the bus encoded in the bus_ID.

- If the bus_ID of the DBC matches the bus_ID of the packet, and if the *tcode* indicates that the packet is a write request for the data quadlet, then the DBC will accept the packet.
- No acknowledge is returned in response to this type of broadcast.

If the 10-bit bus_ID is set to 3FFh and the physical_ID is set to 3Fh, then this is a broadcast to the local bus.

- If the *tcode* indicates that the packet is a write request for the data quadlet, then the DBC will accept the packet.
- No acknowledge is returned in response to this type of broadcast.

A packet addressed to a minimal DBC link controller containing a *tcode* other than 0 or 4 will be acknowledged with an *ack_type_error* response.

Ack_type_error indicates that a field in the request packet header was set to an unsupported or incorrect value, or that an invalid transaction was attempted.

7.2.2 Generating Packets

A minimal implementation of a DBC Link controller must be set to generate one type of packet and transaction code, which is shown in Table 7-2.

Table 7-2. Transaction Code for Packet Generation

Packet	Transaction code
Read response for data quadlet	6h

The read response packet is generated as a concatenated subaction to a read request.

7.3 Retry Code

A minimal DBC Link controller does not support retries; the BUSY_TIMEOUT register is not implemented. The *rt* field in a read response for data quadlet packet must always be retry_X.

7.4 Retries

Retries are not supported in a minimal DBC Link controller; the BUSY_TIMEOUT register is not implemented. A minimal DBC Link controller does not retry a response packet.

7.5 Response Codes

A minimal DBC Link controller must respond to requests using the response codes shown in Table 7-3.

Table 7-3. DBC Link Controller Response Codes

Response code	Response name	Comments
0h	resp_complete	The node has completed the command — no errors.
6h	resp_type_error	A field in the request packet header was set to an unsupported or incorrect value, or an invalid transaction was tried (such as a write to a read-only address).

7.6 Acknowledge Codes

A minimal DBC Link controller must respond to the packets it recognizes with an acknowledge packet. The ack_codes used by a minimal DBC Link controller are listed in Table 7-4. **Table 7-3. DBC Link Controller AcknowledgeCodes**

Acknowledgment code	Acknowledgment name	Comments
1h	ack_complete	The node has successfully accepted the packet. This is sent when the DBC receives a non-broadcast write request for data quadlet (*tcode* = 0) to a valid, writable DBC address.
2h	ack_pending	This is sent in response to a read request for data quadlet if the read targets a valid DBC address. To be followed by a concatenated response packet containing the data quadlet.
Eh	ack_type_error	A field in the request packet header was set to an unsupported or incorrect value, or an invalid transaction was tried. This is sent in response to all requests other than requests with *tcode* values of 0 and 4.

7.7 Physical Interface

The first-generation DBC link controller must interface to a 400 Mbps 1394 PHY. Future generations of DBC link controllers will need to interface to next-generation 1394 PHYs at future 1394 bus speeds (for example, 1394a PHYs, and speeds of 800 Mbps or higher).

7.8 Additional Features

Previous sections detailed the minimum behavior required of the 1394 Link controller section of a DBC. However, not all Link controller implementations can meet the established minimum behavior. If a particular link controller implementation must add extra complexity, then it is the responsibility of the Link controller implementers to account for the extra functionality required by their design, assuring that their design works properly on a 1394 bus.

Examples of additional features required in a non-minimal link controller are listed below. Many more possibilities exist. However, listing all possible cases where a DBC link controller would add extra complexity to its design is beyond the scope of this document.

- Split transactions — If a DBC Link controller implementation can be a requestor on the 1394 bus and uses split transactions, then the SPLIT_TIMEOUT register and extra logic to support split transactions must be included the Link controller design.
- Retries — If the Link controller implementation must issue retries, then the BUSY_TIMEOUT register must be implemented along with the logic necessary to handle the retries.

8 Appendix (Informative)

This section shows an example implementation of a DBC Configuration ROM.

8.1 Example DBC Configuration ROM

The base address for this table is FFFF F000 0000_{16}. The location of the Bus_Info_Block and the Root_Directory are fixed. The location of the Node_Unique_ID leaf and Unit_Directory are implementation-dependent, but their offsets are specified in the root directory.

	Offset				
Bus_Info_Block	400	info_length	crc_length	rom_crc_value	
	404	31_{16}	33_{16}	39_{16}	34_{16}
	408	Node_Options			
	40C	node_vendor_id			chip_id_hi
	410	chip_id_lo			
Root_Directory	414	length		CRC	
	418	03_{16}	module_vendor_ID		
	41C	$0C_{16}$	node_capabilities		
	420	$8D_{16}$	indirect_offset		
	424	$D1_{16}$	unit_directory_offset		
Unit_Directory	434	length		CRC	
	438	12_{16}	unit_spec_ID		
	43C	13_{16}	unit_sw_version		
Node_Unique_ID leaf	428	length		CRC	
	42C	node_vendor_id			chip_id_hi
	430	chip_id_lo			

9 Appendix (Informative)

This section shows an example of the logic that can be performed in the DBC when a hardware removal request button is implemented on the Device Bay subsystem
The DBC logic, or rules, are shown in the following table. For more information, see section 6.6.6.1 of the Device Bay Specification.

State Transition	**Cause**
Any state to Bay Empty	When both the 1394PRSN# and USBPRSN# pins are not asserted low.
Bay Empty to Device Debounced	When either the 1394PRSN# pin or the USBPRSN# pin is asserted low.
Device Debounce to device inserted	Debounce timer expired
Device Inserted to Device Enabled	When SetFeature request is received with *wValue* set to REQUEST_DEVICE_ENABLED_STATE
Any state (other than Bay Empty) to Removal Requested	When removal request button is pushed.
Removal Requested to Removal Allowed	When Software-controlled interlock is disengaged AND Vid power is disabled AND Removal request interrupt is cleared. AND REQUEST_REMOVAL_ALLOWED_STATE

10 Appendix (Informative)

This section shows the role of a USB-based DBC in the sequence of steps that carry out insertion of a device in a bay and removal of a device from a bay.

10.1 *USB-Based DBC Insertion Task Sequence Table*

This scenario begins when a user inserts a USB, 1394, or combination USB/1394 device into a bay that is controlled by a USB-based DBC. The scenario ends when the device or drivers are loaded for the device and the device is powered-up, configured, and operational. This section parses the tasks carried out by the Device Bay mechanical features, the DBC, and the OS to accomplish device insertion. The following Table shows the role of the USB-based DBC in this insertion scenario.

USB-Based DBC Insertion Task Sequence Table

Coordinating the Bay	Mechanical Features	DBC/USB Controller	OS	Coordinating the UI
1-Device is inserted into the bay *x*.				
	2-A shelf provides rough alignment.			
	3-Device Bay connector fine-tunes the alignment.			
	4-Device seats into the connector.			
	5-Connector presence pin(s) asserted.			
		6-Sets 1394PRSN_STS, bit 9, or USBPRSN_STS, bit 8, in the Bay status bitmap (if compound device, sets both bits).		
		7 – Enters Device Debounce state and sets BAY_ST, bits 14-12, in the Bay status bitmap to 101.		
	8 – If present, the hardware bay status indicator is set to indicate Device Debounce.			
		9-After Debounce timer has expired, sets DEVSTSCHG, bit 10, in the Bay status bitmap..		
		10- Sets BAY_ST, bits 14-12, in the Bay status bitmap to 001b.		
		11-If DEVSTSCH_EN is set, see bit 2 of the Bay status bitmap, an interrupt is generated internal to the DBC.		
		12-A USB INTERRUPT TRANSACTION occurs between the DBC Class Driver and the USB DBC controller.		

USB-Based DBC Insertion Task Sequence Table

Coordinating the Bay	Mechanical Features	DBC/USB Controller	OS	Coordinating the UI
			13-Sends a GET_STATUS request and determines the cause of the interrupt by detecting that DEVSTSCHG bit is asserted in the Bay status bitmap.	
				14-Bay status indicator set to indicate Device Inserted State.
			15-Sends CLEAR_FEATURE_C_DEVICE_STATUS_CHANGE.	
		16 – Clears DEVSTSCHG, bit 10, in the Bay status bitmap		
			17-Sends a feature request, SET_FEATURE_LOCK_CTL to engage the software-controlled interlock and physically lock the device into place.	
		18-Engages the software-controlled interlock and sets LOCK_CTL, bit 7, in the Bay status bitmap.		
	19-Software-controlled interlocks are engaged.			
			20-Determines that 1.5W is available to turn on the V_{id} power rail in bay x.	
			21-Sends Feature Request, SET_FEATURE_ENABLE_VID_POWER.	
		22-Turns on V_{id} power rails to bay x, and sets PWR_CTL, bit 0, in the Bay status bitmap.		
	23-V_{id} power flows to the			

USB-Based DBC Insertion Task Sequence Table

Coordinating the Bay	Mechanical Features	DBC/USB Controller	OS	Coordinating the UI
	device in bay x			
			24-Device appears on native bus; native bus enumerates the device.	
			25-Identifies device.	
			26-Determines that adequate power exists and enables V_{op}.	
			27-If bay supports VOP switching, sends SET_FEATURE_ _ENABLE_VOP_POWER	
			28-Sends SET_FEATURE_REQUEST_DEVICE_ENABLED_STATE.	
		29-Sets BAY_ST, bits 14-12, in the Bay status bitmap to 010b.		
	30a-If present, the hardware bay status indicator is set to indicate Device Enabled.			30b-If active, the UI bay status indicator is set to indicate Device Enabled.
31-User sees the device is ready.				

10.2 Button-Initiated Removal Request, USB-Based DBC

This section parses the tasks carried out by the Device Bay mechanical features, the DBC, the USB root hub, and the OS to accomplish device removal after the user presses the removal request button at bay x. Note that the removal request button is an option on a Device Bay. If there is no removal request button on a bay, the user must initiate a device removal request from UI displayed by the OS (for more information about this, see the next topic).

USB-Based DBC Button Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
1-User presses removal request button				
	2-Removal Request signal is asserted.			
		3-Sets REMREQ_STS, bit 11, in the Bay status bitmap.		
		4-Bay state BAY_ST, bits 14-12, in the Bay status bitmap change to 011b to indicate a removal requested.		
	5a-If present, the hardware bay status indicator is set to indicate Device Removal Requested.			5b-If active, the UI bay status indicator is set to indicate Device Removal Requested.
		6-If Removal Request interrupts are enabled, see bit 3 of the Bay status bitmap, an interrupt is generated internal to the DBC.		
		7-A USB INTERRUPT TRANSACTION occurs between the DBC class driver and the USB DBC controller.		
		8-The USB root hub controller generates a system interrupt.		
			9-Sends a GET_STATUS request and determines the cause of the interrupt by detecting that the REMREQ_STS bit is asserted in Bay status bitmap.	

USB-Based DBC Button Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
			10-Sends a Feature Request, CLEAR_FEATURE_C_REMOVE_REQUEST to the DBC via the USB root hub controller to clear the REMREQ_STS, bit 11, in the Bay status bitmap.	
			11-If "appropriate," place device in a logical "off" state using native bus driver (for example, notify apps, etc.).	
	12-Device stops using V_{op}.			
			13-If the bay supports Vop switching as indicated by bit 5 in the Subsystem descriptor, sends a Feature Request, CLEAR_FEATURE_ENABLE_VOP_POWER.	
			14-If possible, unloads any appropriate device drivers.	
			15-Sends an Feature Request, CLEAR_FEATURE_ENABLE_VID_POWEER, to the DBC via the USB root hub controller to disable V_{id} power.	
		16-Disables the V_{id} power rail in bay x, and clears PWR_CTL, bit 0, in the Bay status bitmap.		
	17-V_{id} is removed from the device.			
			18-Waits the time interval specified by the	

USB-Based DBC Button Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
			device.	
			19-Sends a Feature Request, CLEAR_FEATURE_LOCK_CTL, to the DBC via the USB root hub controller to unlock the software-controlled interlock.	
		20-Disengages the software-controlled interlock and clears LOCK_CTL, bit 7, in the Bay status bitmap.		
	21-Software-controlled interlocks are disengaged.			
			22-If any bays have a security lock as indicated by bit 4 in the Subsystem descriptor; sends a GET_STATUS request to determine the state of SL_STS, bit 15, in the Bay status bitmap. If bit not set, skips ahead to step 26.	
			23-Prompts user that before device can be removed, user must disengage security lock.	
				24-User responds to prompt.
			25-Determines if security lock is disengaged; sends a GET_STATUS request to determine if SL_STS, bit 15, in the Bay status bitmap is still set. If	

USB-Based DBC Button Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
			bit is set, go back to step 23.	
			26-Sends a Feature Request, SET_FEATURE_REQUEST_REMOVAL_ALLOWED, to the DBC via the USB root hub controller to request bay state indicator change to Device Removal Allowed (this transaction writes 100b into the BAY_STREQ, bits 6-4, of the Bay status bitmap).	
		27-Sets the BAY_ST, bits 14-12, in the Bay status bitmap to 100b to indicate the Removal Allowed state.		
	28a-If present, the hardware bay status indicator is set to indicate Device Removal Allowed.			28b-If active, the UI bay status indicator is set to indicate Device Removal Allowed.
29-User realizes it is safe to remove the device.				
	30-If present, eject mechanism ejects device for user			
31-User removes device.				
	32-Connector presence pin(s) deasserted.			
		33-Clears 1394PRSN_STS, bit 9, or USBPRSN_STS, bit 8, in the Bay status bitmap (if compound device, clears both bits).		
		34-If REMOVAL_WAKE_EVENT_ENABLE is set (see bit 1 in		

USB-Based DBC Button Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
		the Bay status bitmap) DEVSTSCHG, bit 10, in the Bay status bitmap is set.		
		35-Sets the BAY_ST, bits 14-12, in the Bay status bitmap to 000b to indicate Bay Empty.		
		36-If Device Status Change and Removal Wake interrupts are enabled, see bits 2 and 1 of the Bay status bitmap, an interrupt is generated internal to the DBC.		
		37-An USB INTERRUPT TRANSACTION occurs between the DBC and the USB root hub controller.		
		38-The USB root hub controller generates a system interrupt.		
			39-Receives the system interrupt.	
			40-Sends a GET_STATUS to each bay to determines the bay location of the interrupt by reading the DEVSTSCHG bit in each Bay status bitmap until it finds one that is set.	
			41-Determines the cause of the interrupt by determining both the 1394PRSN_STS and USBPRSN_STS bits are cleared, so device must have	

USB-Based DBC Button Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
			been removed.	
			42-Sends a FEATURE_REQUEST, CLEAR_FEATURE_C_DEVICE_STATUS_CHANGE, to the DBC via the USB root hub controller to clear the "sticky" DEVSTSCHNG, bit 10, in the Bay status bitmap.	
	43a-If present, the hardware bay status indicator is set to indicate Bay Empty.			43b-If active, the UI bay status indicator is set to indicate Bay Empty.
				44-User gets closure on the request.

10.3 UI-Initiated Removal Request, USB-Based DBC

This section parses the tasks carried out by the Device Bay mechanical features, the DBC, the USB root hub, and the OS to accomplish device removal after the user initiates a device removal request from the UI displayed by the OS.

USB-Based DBC UI Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
				1-User initiates a device removal request.
			2-Uses information returned by the UI to determine which bay the user wants to remove the device from (bay x), then sends a Feature Request, SET_FEATURE_REMOVAL_REQUESTED_STATE, to the DBC via the USB root hub controller to set the bay to the Removal Requested state (this transaction writes 011b to BAY_STREQ, bits 6-4, in the Bay Status bitmpa).	
		3-Sets BAY_ST, bits 14-12, in the Bay status bitmap to 011b to indicate Removal Requested.		
	4(a)-If present, the hardware bay status indicator is set to indicate Device Removal Requested.			4(b)-If hardware bay status indicator is not present, the UI bay status indicator is set to indicate Device Removal Requested.
5-User realizes system has begun process of removing device from bay x.				
			6-If “appropriate”, place device in a logical “off” state using native bus driver (for example, notify apps, etc.).	
	7-Device stops using V_{op}.			

USB-Based DBC UI Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
			8-If the bay supports Vop switching as indicated by bit 5 in the Subsystem descriptor, sends a Feature Request, CLEAR_FEATURE_ENABLE_VOP_POWER.	
			9-Unloads any appropriate device drivers.	
			10-Sends a Feature Request, CLEAR_FEATURE_ENABLE_VID_POWER, to the DBC via the USB root hub controller to turn off V_{id} power in bay x.	
		11-Disables V_{id} power rail in bay x and clears PWR_CTL, bit 0, in the Bay status bitmap		
	12-V_{id} is removed from device.			
			13-Waits the time interval specified by the device.	
			14-Sends a Feature Request, CLEAR_FEATURE_LOCK_CTL, to the DBC via the USB root hub controller to unlock the software-controlled interlock.	
		15-Disables the software-controlled interlock and clears LOCK_CTL, bit 7, in the Bay status bitmap.		
	16-Software-controlled interlocks are disabled.			

USB-Based DBC UI Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
			17-If any bays have a security lock as indicated by bit 4 in the Subsystem descriptor; sends a GET_STATUS request to determine the state of SL_STS, bit 15, in the Bay status bitmap. If bit not set, skips ahead to step 21.	
			18-Prompts user that before device can be removed, user must disengage security lock.	
				19-User responds to prompt.
			20- Determines if security lock is disengaged; sends a GET_STATUS request to determine if SL_STS, bit 15, in the Bay status bitmap is still set. If bit is set, go back to step 18.	
			21-Sends a Feature Request, SET_FEATURE_REQUEST_REMOVAL_ALLOWED_STATE, to the DBC via the USB root hub controller to change the bay indicator state to Device Removal Allowed (this sets BAY_STREQ, bits 6-4, in the Bay status bitmap to 100b.	
		22-Sets the BAY_ST, bits 14-12, in the Bay status bitmap to 100b to indicate Device Removal		

USB-Based DBC UI Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
		Allowed.		
	23(a)-If present, the hardware bay status indicator is set to indicate Device Removal Allowed.			23(b)-If active, the UI bay status indicator is set to indicate Device Removal Allowed.
			24-Pops up a UI to inform user that it is safe to remove the device.	
25-User realizes device can be removed.				
				26-User realizes it is safe to remove the device.
	27-If present, eject mechanism ejects device for user			
28-User removes device.				
	29-Connector presence pin(s) deasserted.			
		30- Clears 1394PRSN_STS, bit 9, or USBPRSN_STS, bit 8, in the Bay status bitmap (if compound device, clears both bits).		
		31- If REMOVAL_WAKE_EVENT_ENABLE is set (see bit 1 of the Bay status bitmap), sets DEVSTSCHG, bit 10, in the Bay status bitmap.		
		32-Sets the BAY_ST, bits 14-12, in the Bay status bitmap to 000b to indicate Bay Empty.		
		33-If Device Status Change and Removal Wake Enable interrupts are		

USB-Based DBC UI Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
		enabled, see bit 2 and 1 of the Bay status bitmap, an interrupt is generated internal to the DBC.		
		34-A USB INTERRUPT TRANSACTION occurs between the DBC and the USB root hub controller.		
		35-The USB root hub controller generates a system interrupt.		
			36-Sends a GET_STATUS request for each bay to determine the bay location of the interrupt by reading the DEVSTSCHG bit in each Bay status bitmap until it finds one that is set.	
			37-Determines the cause of the interrupt by determining both the 1394PRSN_STS, bit 9, and USBPRSN_STS, bit 8, in the Bay status bitmap are cleared, so device must have been removed.	
			38-Sends a Feature Request, CLEAR_FEATURE_C_DEVICE_STATUS_CHANGE, to the DBC via the USB root hub controller to clear the “sticky” DEVSTSCHNG, bit 10, in the Bay status bitmap.	
	39(a)-If present, the hardware bay status indicator is set to		39(b)-If active, the UI bay status indicator is set to indicate Bay	

USB-Based DBC UI Initiated Removal Request Task Sequence

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
	indicate Bay Empty.		Empty.	
				440-User gets closure on the request.

10.4 System Power On Task Sequence Tables

As the system powers up, the DBC resets and the PWR_EN and LOCK_EN control signals are negated. Therefore, V_{id} will not be applied to any present devices and the software-controlled interlocks will not be engaged. (Device retention mechanisms or optional security mechanisms may be engaged.) After the OS has loaded, the OS will read the DBC Bay status bitmaps, to determine if devices are present, and proceed through the appropriate previously defined device insertion sequence.
A platform provider may chose to support booting from a Device Bay device. In this case, the system BIOS must have the ability to find and communicate with the DBC, identify possible boot devices via their native bus, (1394 or USB), and load the OS from a Device Bay device. A possible sequence is described in the following section.

10.4.1 System Power On Task Sequence Tables

This section parses the tasks carried out by the Device Bay mechanical features, the DBC, the USB root hub, the system BIOS, and the OS when the system is powered up.

Coordinating the System and Bay	Mechanical Features	DBC/USB Root Hub Controller	OS / BIOS	Coordinating the UI
1-User initiates a system power-up.				
		2-LOCK_EN and PWR_EN for all bays are negated. Interrupt enables are negated.		
		3-Sets 1394PRSN_STS, bit 9, or USBPRSN, bit 8, in the Bay status bitmap (if compound device, sets both bits) for each bay with a device present		
		[NOTE: No interrupt is generated from devices already present during the power-up sequence.]		
			4-BIOS determines the number of bays in the system and the bay types from the DBC Subsystem descriptor.	
			5-BIOS determines the total Device Bay sub-system power and thermal capabilities from the DBC Subsystem descriptor.	
			6-BIOS determines the maximum number of devices that can be supported. [Note: Possible power budgeting software agents are not likely active yet.]	
			7-BIOS determines if the system OS is supported via a Device Bay device. If not, once the OS is loaded, it will control the Device Bay sub-system. If yes, continue.	
			8-BIOS sends a GET_STATUS request to the DBC to determine the state of 1394PRSN_STS, bit	

Coordinating the System and Bay	Mechanical Features	DBC/USB Root Hub Controller	OS / BIOS	Coordinating the UI
			9, and USBPRSN_STS, bit 8, of the bays; if one (or both) are set in any bays, a device(s) are present.	
			9-BIOS sends a Feature Request, SET_FEATURE_LOCK_CTL, to the DBC via the USB root hub controller to physically lock the first present device into place.	
		10-Enables the software-controlled interlock and sets LOCK_CTL, bit 7, in the Bay status bitmap.		
	11-Software-controlled interlocks are enabled.			
			12-BIOS sends a Feature Request, SET_FEATURE_ENABLE_VID_POWER, to the DBC via the USB root hub controller to enable V_{id} power to the device.	
		13-Enables V_{id} power rails to bay x and sets PWR_CTL, bit 0, in the Bay status bitmap.		
	14-V_{id} power flows to the device in bay x			
			15-Device appears on native bus. BIOS identifies the device to determine if it can load the OS from this device type. If yes, skip to #23.	
			16-Sends a Feature Request, CLEAR_FEATURE_ENABLE_VID_POWER, to the DBC via the USB root hub controller to disable V_{id} power to bay x.	

Coordinating the System and Bay	Mechanical Features	DBC/USB Root Hub Controller	OS / BIOS	Coordinating the UI
		17-Disables V_{id} power rail to bay x and clears PWR_CTL, bit 0, in the Bay status bitmap.		
	18-V_{id} is removed the device.			
			19-Return to #9 and continue looking for a possible boot device.	
			20-BIOS sends a Feature Request, SET_FEATURE_REQUEST_DEVICE_ENABLED_STATE, to the DBC via the USB root hub controller to request bay state change to Device Enabled (this transaction writes 010b to BAY_STREQ, bits 6-4, in the Bay Status bitmap).	
		21-Sets BAY_ST, bits 14-12, in the Bay status bitmap to 010b to indicate Device Enabled.		
	22-If present, the hardware status indicator is set to indicate device enabled.			
			23-If the bay supports Vop switching as indicated by bit 5 in the Subsystem descriptor, sends a Feature Request, SET_FEATURE_ENABLE_VOP_POWER. If not skip to #25.	
		24-If subsystem Vop power switching is supported, enables the subsystem Vop power rails.		
			25-BIOS indicates via the native bus that the device can enable V_{op}.	
			26-BIOS determine if the OS is present on the	

Coordinating the System and Bay	Mechanical Features	DBC/USB Root Hub Controller	OS / BIOS	Coordinating the UI
			device. If yes, skip to #38.	
			27-BIOS indicates via the native bus that the device must disable V_{op}.	
			28-If the bay supports Vop switching as indicated by bit 5 in the Subsystem descriptor, sends a Feature Request, CLEAR_FEATURE_ENABLE_VOP_POWER. If not skip to #30.	
		29-If Vop power switching is supported by the Subsystem, disables the subsystem Vop power rail.		
			30-Sends a Feature Request, CLEAR_FEATURE_ENABLE_VID_POWER, to the DBC via the USB root hub controller to turn off V_{id} power in bay x.	
		31-Disables V_{id} power rail in bay x and clears PWR_CTL, bit 0, in the Bay status bitmap.		
	32-V_{id} is removed from device.			
			33-Waits the time interval sp-ecified by the device.	
			34-Sends a Feature Request, SET_FEATURE_REQUEST_DEVICE_INSERTED_STATE, to the DBC via the USB root hub controller to return the bay indicator state to Device is inserted but not ready (this sets BAY_STREQ, bits 6-4, in the Bay status bitmap).	
		35-Sets the BAY_ST, bits 14-12, in the Bay status		

Coordinating the System and Bay	Mechanical Features	DBC/USB Root Hub Controller	OS / BIOS	Coordinating the UI
		bitmap to 001b to indicate Device Inserted.		
	36-If present, the hardware bay status indicator is set to indicate Device Inserted.			
			37-Return to #9 and find the next device.	
			38-OS is booted and assumes control of the Device Bay sub-system	

10.5 System Power Down Task Sequence Tables

This section parses the tasks carried out by the Device Bay mechanical features, the DBC, the USB root hub, and the OS to power-down the system after the user initiates a shutdown request from the UI displayed by the OS.

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
				1-User initiates a system shutdown request.
			2-If “appropriate,” place device in a logical “off” state using native bus driver (for example, notify apps, etc.).	
	3- Device stops using V_{op}.			
			4- If the bay supports Vop switching as indicated by bit 5 in the Subsystem descriptor, sends a Feature Request, CLEAR_FEATURE_ENABLE_VOP_POWER. If not skip to #6.	
		5-If subsystem Vop power switching is supported, disables the Vop power rails.		
			6-Sends a Feature Request, CLEAR_FEATURE_ENABLE_VID_POWER, to the DBC via the USB root hub controller to disable V_{id} power to each bay.	
		7-Disables V_{id} power rail to all bays and clears the PWR_CTL bits, bit 0, in the Bay status bitmaps.		
	8-V_{id} is removed from devices.			
			9-Waits the longest time interval specified by any device.	
			10-If security lock is	

Coordinating the Bay	Mechanical Features	DBC/USB Root Hub	OS	Coordinating the UI
			supported as indicated by bit 4 in the DBC Subsystem descriptor, sends a GET_STATUS request to each bay to determine the state of the security locks via SL_STS, bit 15, in the Bay status bitmap.	
				11-If necessary, a UI indicating the bays that are locked is displayed.
			12-Sends a Feature Request, SET_FEATURE_REQUEST_REMOVAL_ALLOWED_STATE, to the DBC via the USB root hub controller to change the bay state for each bay to Device Removal Allowed, (provided a security lock is not present and engaged).	
		13- Sets the BAY_ST, bits 14-12, in the Bay status bitmaps to 100b to indicate Device Removal Allowed for each bay, as appropriate.		
14-User realizes devices can be removed.				
			15-Shutdown process completes	
				16-System power-down message is displayed.
17-User may remove power and unlocked devices.				

10.6 Button-Initiated Device Removal Scenario

This scenario begins when a user requests the removal of a device from a bay that is controlled by a USB-based DBC; the user can make a device removal request either through software (for example, by a UI selection) or through hardware (for example, by pressing a removal-request button). The scenario ends when the device has been physically removed from the bay. The following flowchart, Figure 10-2, shows the role of the USB-based DBC in this device removal scenario.
the

Figure 10-2. Device Removal Scenario Flow Chart