Chapter 9
USB Device Framework

9.1 Introduction

A USB device may be divided into three layers. The bottom layer is a bus interface that transmits and receives packets. The middle layer handles routing data between the bus interface and various endpoints on the device. An endpoint is the ultimate consumer or provider of data. It may thought of as a source or sink for data. The top layer is the functionality provided by the serial bus device; for instance, a mouse or ISDN interface.

This chapter describes the common attributes and operations of the middle layer of a USB device. These attributes and operations are used by the function-specific portions of the device to communicate through the bus interface and ultimately with the host.

9.2 USB Device States

A USB device has several possible states. Some of these states are visible to the USB and the host and others are internal to the USB device. This section describes those states.

9.2.1 Visible Device States

This section describes USB device states that are externally visible. Note: USB devices perform a reset operation in response to a Reset request to the upstream port from the host. When reset signaling has completed, the USB device is reset.

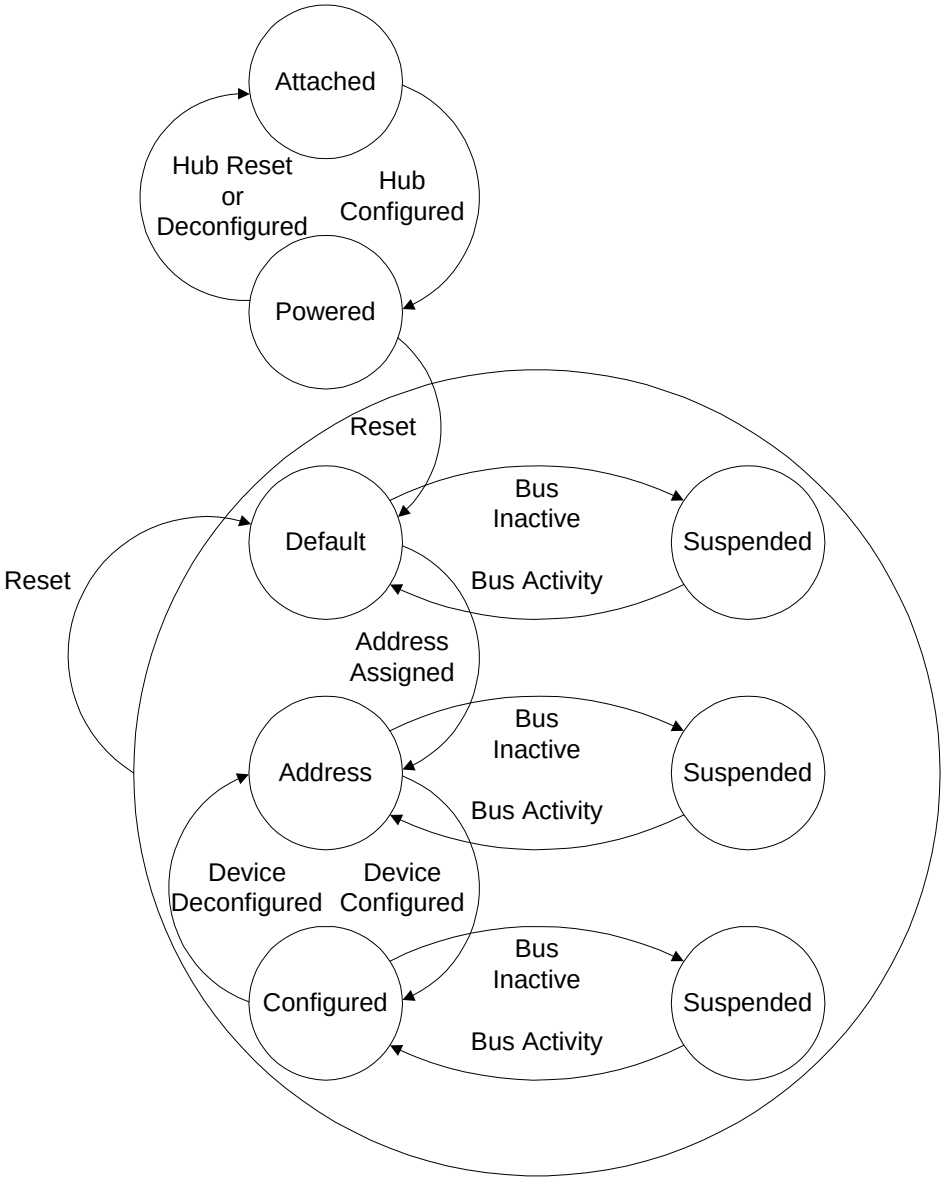

Figure 9-1. Device State Diagram

Table 9-1. Visible Device States

Attached	Powered	Default	Address	Configured	Suspended	State
No	--	--	--	--	--	Device is not attached to USB. Other attributes are not significant.
Yes	No	--	--	--	--	Device is attached to USB, but is not powered. Other attributes are not significant.
Yes	Yes	No	--	--	--	Device is attached to USB and powered, but has not been Reset.
Yes	Yes	Yes	No	--	--	Device is attached to USB and powered and has been Reset, but has not been assigned a unique address. Device responds at the default address.
Yes	Yes	Yes	Yes	No	--	Device is attached to USB, powered, has been Reset, and a unique device address has been assigned. Device is not configured.
Yes	Yes	Yes	Yes	Yes	No	Device is attached to USB, powered, has been Reset, has unique address, is configured, and is not suspended. Host may now use the function provided by the device.
Yes	Yes	Yes	Yes	Yes	Yes	Device is, at minimum, attached to USB, has been reset and is powered at the minimum suspend level, It may also have a unique address and be configured for use. However, since the device is suspended, the host may not use the device's function.

9.2.1.1 Attached

A USB device may be attached or detached from the USB. The state of a USB device when detached from the USB is not defined by this specification. This specification only addresses required operations and attributes once the device is attached.

9.2.1.2 Powered

USB devices may obtain power from an external source and/or from USB through the hub to which they are attached. Externally powered USB devices are termed self-powered. These devices may already be powered before they are attached to the USB. A device may support both self-powered and bus-powered configurations. Some device configurations support either power source. Other device configurations may only be available if the device is externally powered. Devices report their power source capability through the Configuration Descriptor. The current power source is reported as part of a device's status. Devices may change their power source at any time; e.g., from self- to USB-powered. If a configuration is capable of supporting both power modes, then the power maximum reported for that configuration is the maximum the device will draw in either mode. The device must observe this maximum, regardless of its mode. If a configuration supports only one power mode and the power source of the device changes, then the device will loose its current configuration and address and return to the attached state.

A hub port must be powered in order to detect port status changes, including attach and detach. Hubs do not provide any downstream power until they are configured, at which point they will provide power as allowed by their configuration and power source. A USB device must be able to be addressed within a specified time period from when power is initially applied (refer to Chapter 7). After an attachment to a port has been detected, the host shall enable the port, which will also reset the device attached to the port.

9.2.1.3 Default

After the device has been powered, it must not respond to any bus transactions until it has received a reset from the bus. After receiving a reset, the device is then addressable at the default address..

9.2.1.4 Address Assigned

All USB devices use the default address when initially powered or after the device has been reset. Each USB device is assigned a unique address by the host after attachment or after reset. A USB device maintains its assigned address while suspended.

A USB device responds to requests on its default pipe whether the device is currently assigned a unique address or is using the default address.

9.2.1.5 Configured

Before the USB device's function may be used, the device must be configured. From the device's perspective, configuration involves writing a non-zero value to the device configuration register. Configuring a device or changing an alternate setting causes all of the status and configuration values associated with endpoints in the affected interfaces to be set to their default values. This includes setting the data toggle of any endpoint using data toggles to the value DATA0.

9.2.1.6 Suspended

In order to conserve power, USB devices automatically enter the Suspended state when the device has observed no bus traffic for a specified period (refer to Chapter 7). When suspended, the USB device maintains any internal status including its address and configuration.

All devices must suspend if bus activity has not been observed for the length of time specified in Chapter 7. Attached devices must be prepared to suspend at any time they are powered, whether they have been

assigned a non-default address or are configured. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled. This is referred to as selective suspend.

A USB device exits suspend mode when there is bus activity. A USB device may also request the host exit suspend mode or a selective suspend by using electrical signaling to indicate Remote Wakeup. The ability of a device to signal remote wakeup is optional. If a USB device is capable of remote wakeup signaling, the device must support the ability of the host to enable and disable this capability.

9.2.2 Bus Enumeration

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached, the following actions are undertaken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe, see Chapter 11. At this point, the USB device is in the attached state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host issues a port enable and reset command to that port.
4. The hub maintains the reset signal to that port for 10 ms. When the reset signal is released, the port has been enabled and the hub provides 100 mA of bus power to the USB device. The USB device is now in the powered state. All of its registers and state have been reset and it answers to the default address.
5. Before the USB device receives a unique address, it's default pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
6. The host assigns a unique address to the USB device, moving the device to the addressed state. As result of entering the addressed state, the device's default pipe now uses its declared maximum data payload size instead of the default of 8 bytes.
7. The host reads the configuration information from the device by reading each configuration zero to n. This process may take several frames to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of Vbus power described in its configuration descriptor. From the device's point of view it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

9.3 Generic USB Device Operations

All USB devices support a common set of operations. This section describes those operations.

9.3.1 Dynamic Attachment and Removal

USB devices may be attached and removed at any time. The hub that provides the attachment point or port is responsible for reporting any change in the state of the port.

The host enables the hub port where the device is attached upon detection of an attachment, which also has the effect of resetting the device. A reset USB device has the following characteristics:

- Responds to the default USB address
- Is unconfigured
- Is not initially suspended

When a device is removed from a hub port, the host is notified of the removal. The host responds by disabling the hub port where the device was attached.

9.3.2 Address Assignment

When a USB device is attached, the host is responsible for assigning a unique address to the device after the device has been reset by the host and the hub port where the device is attached has been enabled.

9.3.3 Configuration

A USB device must be configured before its function may be used. The host is responsible for configuring a USB device. The host typically requests configuration information from the USB device to determine the device's capabilities.

As part of the configuration process, the host sets the device configuration and, where necessary, sets the maximum packet size for endpoints that require such limitation.

Within a single configuration, a device may support multiple interfaces. An interface is a related set of endpoints that present a single feature or function of the device to the host. The protocol used to communicate with this related set of endpoints and the purpose of each endpoint within the interface may be specified as part of a device class or vendor specific class definition.

In addition, an interface within a configuration may have alternate settings that redefine the number or characteristics of the associated endpoints. If this is the case, the device shall support the Get Interface and Set Interface requests to report or select a specific alternative setting for a specific interface.

Within each configuration, each interface descriptor contains fields that identify the interface number and the alternate setting. Interfaces are numbered from zero to one less than the number of concurrent interfaces supported by the configuration. Alternate settings range from zero to one less than the number of alternate settings for a specific interface. The default setting when a device is initially configured is alternate setting zero.

In support of adaptive device drivers that are capable of managing a related group of USB devices, the device and interface descriptors contain Class, SubClass, and Protocol fields. These fields are used to identify the function(s) provided by a USB device and the protocols used to communicate with the function(s) on the device. A Class code is assigned to a related class of devices that has been defined as a part of the USB specification. A class of devices is further subdivided into subclasses and within a class or subclass a protocol code defines how host software communicates with the device.

9.3.4 Data Transfer

Data may be transferred between a USB device endpoint and the host in one of four ways. Refer to Chapter 5 for the definition of the four types of transfers. Some endpoints may be capable of different types of data transfers. However, once configured, a USB device endpoint uses only one data transfer method.

9.3.5 Power Management

Power management on USB devices involves the issues described in the following sections.

9.3.5.1 Power Budgeting

For bus-powered devices, power is a limited resource. When a host detects the attachment of a bus-powered USB device, the host needs to evaluate the power requirements of the device. If USB device power requirements exceed available power, the device is not configured.

No USB device may require more than 100 mA when first attached. A configured bus-powered USB device attached to a self-powered hub may use up to 500 mA; however, some ports may not be able supply this much power and thus the device will not be usable.

All USB devices must support a suspended mode that requires less than 500 μa. A USB device automatically suspends when the bus is inactive, as previously described.

9.3.5.2 Remote Wakeup

Remote Wakeup allows a suspended USB device to signal a host that may also be suspended. This notifies the host that it should resume from its suspended mode, if necessary, and service the external event that triggered the suspended USB device to signal the host. A USB device reports its ability to support Remote Wakeup in a Configuration Descriptor. If a device supports Remote Wakeup, it must also allow the capability to be enabled and disabled using the standard USB requests.

Remote Wakeup is accomplished using electrical signaling described elsewhere in this document.

9.4 USB Device Requests

All USB devices respond to requests from the host on the device's default pipe. These requests are made using control transfers. The request and the request's parameters are sent to the device in the setup packet. The host is responsible for establishing the values passed in the following fields. Every setup packet has eight bytes, used as follows:

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bit-map	Characteristics of Request D7 Data xfer direction 0 = Host to device 1 = Device to host D6..5 Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4..0 Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4..31 = Reserved
1	bRequest	1	Value	Specific Request (refer to Table 9-2)
2	wValue	2	Value	Word-sized field that varies according to request
4	wIndex	2	Index or Offset	Word sized field that varies according to request - typically used to pass an index or offset
6	wLength	2	Count	Number of bytes to transfer if there is a data phase

9.4.1 bmRequestType

This bit-mapped field identifies the characteristics of the specific request. In particular, this field identifies the direction of data transfer in the second phase of the control transfer. The state of the direction bit is ignored if the *wLength* field is zero, signifying there is no data phase.

The USB Specification defines a series of Standard requests that all devices must support. In addition, a device class may define additional requests. A device vendor may also define requests supported by the device.

Requests may be directed to the device, an interface on the device, or a specific endpoint on a device. This field also specifies the intended recipient of the request. When an interface or endpoint is specified, the *wIndex* field identifies the interface or endpoint.

9.4.2 bRequest

This field specifies the particular request. The *Type* bits in the *bmRequestType* field modify the meaning of this field. This specification only defines values for the bRequest field when the bits are reset to zero indicating a Standard request (refer to Table 9-2).

9.4.3 wValue

The contents of this field vary according to the request. It is used to pass a parameter to the device specific to the request.

9.4.4 wIndex

The contents of this field vary according to the request. It is used to pass a parameter to the device specific to the request.

9.4.5 wLength

This field specifies the length of the data transferred during the second phase of the control transfer. The direction of data transfer (host to device or device to host) is indicated by the *Direction* bit of the *bRequestType* field. If this field is zero, there is no data transfer phase.

9.5 Standard Device Requests

This section describes the standard device requests defined for all USB devices (refer to Table 9-2).

USB devices must respond to standard device requests whether the device has been assigned a non-default address or the device is currently configured.

Table 9-2. Standard Device Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Table 9-3. Standard Request Codes

bRequest	Value
GET_STATUS	0
CLEAR_FEATURE	1
reserved for future use	2
SET_FEATURE	3
reserved for future use	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

Table 9-4. Descriptor Types

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5

Feature selectors are used when enabling or setting features, such as remote wakeup, specific to a device, interface or endpoint. The values for the feature selectors are given below.

Feature Selector	Recipient	Value
DEVICE_REMOTE_WAKEUP	device	1
ENDPOINT_STALL	endpoint	0

If an unsupported or invalid request is made to a USB device, the device responds by indicating a stall condition on the pipe used for the request. Control pipes, including the default pipe, must accept a setup transaction even if they are stalled. The ClearStall request is used to clear a stalled pipe. After the stall condition is cleared by the host, system software continues normal accesses to the control pipe. If for any reason, the device becomes unable to communicate via its default pipe due to an error condition, the device must be reset to clear the condition and restart the default pipe.

9.5.1 Clear Feature

This request is used to clear or disable a specific feature

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None

Feature selector values in *wValue* must be appropriate to the recipient. Only device feature selector values may be used when the recipient is a device, only interface feature selector values may be used when the recipient is an interface, and only endpoint feature selector values may be used when the recipient is an endpoint.

Refer to Section 9.5 for a definition of which feature selector values are defined for which recipients.

A ClearFeature request that references a feature that can not be cleared or that does not exist will cause a stall.

9.5.2 Get Configuration

This request returns the current device configuration.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value

If the returned value is zero, the device is not configured.

9.5.3 Get Descriptor

This request returns the specified descriptor if the descriptor exists.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID (refer to Section 9.7.5)	Descriptor Length	Descriptor

The *wValue* field specifies the descriptor type in the high byte and the descriptor index in the low byte (refer to Table 9-4). The *wIndex* field specifies the Language ID for string descriptors or is reset to zero for other descriptors. The *wLength* field specifies the number of bytes to return. If the descriptor is longer than the *wLength* field, only the initial bytes of the descriptor are returned. If the descriptor is shorter than the *wLength* field, the device indicates the end of the control transfer by sending short packet when further data

is requested. A short packet is defined as a packet shorter than the maximum payload size or a NULL data packet (refer to Chapter 5).

The Standard request to a device supports three types of descriptors: DEVICE, CONFIGURATION, and STRING. A request for a configuration descriptor returns the configuration descriptor, all interface descriptors, and endpoint descriptors for all of the interfaces in a single request. The first interface descriptor immediately follows the configuration descriptor. The endpoint descriptors for the first interface follow the first interface descriptor. If there are additional interfaces, their interface descriptor and endpoint descriptors follow the first interface's endpoint descriptors.

All devices must provide a device descriptor and at least one configuration descriptor. If a device does not support a requested descriptor, it responds by stalling the pipe used for the request. A non-zero value as the first byte of a descriptor indicates the buffer contains a valid descriptor.

9.5.4 Get Interface

This request returns the selected alternate setting for the specified interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Setting

Some USB devices have configurations with interfaces that have mutually exclusive settings. This request allows the host to determine the currently selected alternative setting.

9.5.5 Get Status

This request returns status for the specified recipient.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status

The Recipient bits of the *bRequestType* field specify the desired recipient. The data returned is the current status of the specified recipient.

A GetStatus request to a device returns the following information in little-endian order:

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)						Remote Wakeup	Self Powered
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

The *Self Powered* field indicates whether the device is currently bus-powered or self-powered. If D0 is reset to zero, the device is bus-powered. If D0 is set to one, the device is self-powered. The *Self Powered* field may not be changed by the SetFeature or ClearFeature requests.

The *Remote Wakeup* field indicates whether the device is currently enabled to request remote wakeup. The default mode for devices which support remote wakeup is disabled. If D1 is reset to zero, the ability of the device to signal remote wakeup is disabled. If D1 is set to one, the ability of the device to signal remote wakeup is enabled. The *Remote Wakeup* field can be modified by the SetFeature and ClearFeature requests using the DEVICE_REMOTE_WAKEUP feature selector. This field is reset to zero when the device is reset.

A GetStatus request to an interface returns the following information in little-endian order:

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

If the request is made to an endpoint, then the endpoint must be specified in the *wIndex* field. The upper byte of *xIndex* is reset to zero and the lower byte contains the endpoint number as follows:

D7	D6	D5	D4	D3	D2	D1	D0
Direction	Reserved (Reset to zero)			Endpoint Number			

For IN endpoints, D7 is set to one. For OUT endpoints, D7 is reset to zero.

A GetStatus request to an endpoint returns the following information:

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							Stall
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

If the endpoint is currently stalled, the *Stall* field is set to one. Otherwise the *Stall* field is reset to zero. The *Stall* field may be changed with the SetFeature and ClearFeature requests with the ENDPOINT_STALL feature selector. When set by the SetFeature request, the endpoint exhibits the same stall behavior as if the field had been set by a hardware condition. If the condition causing a stall has been removed, clearing the stall field results in the endpoint no longer returning a stall status. For this endpoints using a data toggle, clearing a stalled endpoint results in the data toggle being reinitialized to DATA0. This field is reset to zero after either the SetConfiguration or SetInterface request.

9.5.6 Set Address

This request sets the device address for all future device accesses.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None

The *wValue* field specifies the device address to use for all subsequent accesses.

As noted elsewhere, requests actually may result in up to three stages. In the first stage, the setup packet is sent to the device. In the optional second stage, data is transferred between the host and the device. In the final stage, status is transferred between the host and the device. The direction of data and status transfer depends on whether the host is sending data to the device or the device is sending data to the host. The status stage is always in the opposite direction of the data stage. If there is no data stage, the status stage is from the device to the host.

Stages after the initial setup packet assume the same device address as the setup packet. The USB device does not change its device address until after the status stage of this request is completed successfully. Note that this is a difference between this request and all other requests. For all other requests, the operation indicated must be completed before the status stage.

9.5.7 Set Configuration

This request sets the device configuration.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None

The *wValue* field specifies the desired configuration. This value must be zero or match a configuration value from a Configuration Descriptor. If the value is zero, the device is placed in its unconfigured state.

9.5.8 Set Descriptor

This request is optional. If a device supports this request, existing descriptors may be updated or new descriptors may be added.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Language ID (refer to Section 9.7.5)	Descriptor Length	Descriptor

The *wValue* field specifies the descriptor type in the high byte and the descriptor index in the low byte (refer to Table 9-4). The *wIndex* field specifies the Language ID for string descriptors or is reset to zero for other descriptors. The *wLength* field specifies the number of bytes to transfer from the host to the device.

9.5.9 Set Feature

This request is used to set or enable a specific feature.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None

Feature selector values in *wValue* must be appropriate to the recipient. Only device feature selector values may be used when the recipient is a device; only interface feature selector values may be used when the recipient is an interface, and only endpoint feature selector values may be used when the recipient is an endpoint.

Refer to Section 9.5 for a definition of which feature selector values are defined for which recipients. A SetFeature request that references a feature that cannot be set or that does not exist causes a stall.

9.5.10 Set Interface

This request allows the host to select an alternate setting for the specified interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000001B	SET_INTERFACE	Alternative Setting	Interface	Zero	None

Some USB devices have configurations with interfaces that have mutually exclusive settings. This request allows the host to select the desired alternate setting.

9.5.11 Synch Frame

This request is used to set and then report an endpoint's synchronization frame.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

When an endpoint supports isochronous transfers, the endpoint may also require per frame transfers to vary in size according to a specific pattern. The host and the endpoint must agree on which frame the repeating pattern begins. This request causes the endpoint to begin monitoring the SOF frame number to track the position of a given frame in its pattern. The number of the frame in which the pattern began is returned to the host. This frame number is the one conveyed to the endpoint by the last SOF prior to the first frame of the pattern.

This value is only used for isochronous data transfers using implicit pattern synchronization. If the endpoint is not isochronous or is not using this method, this request is not supported by the endpoint and returns stall.

The starting frame is reset to zero by a device reset or by configuring the endpoint either via a SetConfiguration or SetInterface request.

9.6 Descriptors

USB devices report their attributes using descriptors. A descriptor is a data structure with a defined format. Each descriptor begins with a byte-wide field that contains the total number of bytes in the descriptor followed by a byte-wide field that identifies the descriptor type.

Using descriptors allows concise storage of the attributes of individual configurations because each configuration may reuse descriptors or portions of descriptors from other configurations that have the same characteristics. In this manner, the descriptors resemble individual data records in a relational database.

Where appropriate, descriptors contain references to string descriptors that provide displayable information describing a descriptor in human-readable form. The inclusion of string descriptors is optional. However, the reference fields within descriptors are mandatory. If a device does not support string descriptors, string reference fields must be reset to zero to indicate no string descriptor is available.

If a descriptor returns with a value in its length field that is less than defined by this specification, the descriptor is invalid and should be rejected by the host. If the descriptor returns with a value in its length field that is greater than defined by this specification, the extra bytes are ignored by the host, but the next descriptor is located using the length returned rather than the length expected.

Class and vendor specific descriptors may be returned in one of two ways. Class and vendor specific descriptors that are related to standard descriptors are returned in the same data buffer as the standard descriptor immediately following the related standard descriptor.

If, for example, a class or vendor specific descriptor is related to an interface descriptor, the related class or vendor specific descriptor is placed between the interface descriptor and the interface's endpoint descriptors in the buffer returned in response to a GET_CONFIGURATION_DESCRIPTOR request. The length of a standard descriptor is not increased to accomodate device class or vendor specific extensions. Class or vendor specific descriptors follow the same format as standard descriptors with the length and type fields as the first two bytes of the specific descriptor.

Class or vendor specific descriptors that are not related to a standard descriptor are returned using class or vendor specific requests.

9.7 Standard USB Descriptor Definitions

9.7.1 Device

A device descriptor describes general information about a USB device. It includes information that applies globally to the device and all of the device's configurations. A USB device has only one device descriptor.

All USB devices have an endpoint zero used by the default pipe. The maximum packet size of a device's endpoint zero is described in the device descriptor. Endpoints specific to a configuration and its interface(s) are described in the configuration descriptor. A configuration and its interface(s) do not include an endpoint descriptor for endpoint zero. Other than the maximum packet size, the characteristics of endpoint zero are defined by this specification and are the same for all USB devices.

The *bNumConfigurations* field identifies the number of configurations the device supports.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	DEVICE Descriptor Type
2	bcdUSB	2	BCD	USB Specification Release Number in Binary-Coded Decimal (i.e. 2.10 is 0x210). This field identifies the release of the USB Specification that the device and its descriptors are compliant with.
4	bDeviceClass	1	Class	Class code (assigned by USB). If this field is reset to zero (0), each interface within a configuration specifies its own class information and the various interfaces operate independently. If this field is set to a value between one (1) and 0xFE (254), the device supports different class specifications on different interfaces and the interfaces may not operate independently. This value identifies the class definition used for the aggregate interfaces. (For example, a CD-ROM device with audio and digital data interfaces that require transport control to eject CDs or start them spinning.) If this field is set to 0xFF, the device class is vendor specific.

Offset	Field	Size	Value	Description
5	bDeviceSubClass	1	SubClass	Subclass code (assigned by USB). These codes are qualified by the value of the *bDeviceClass* field. If the *bDeviceClass* field is reset to zero (0), this field must also be reset to zero (0). If the *bDeviceClass* field is not set to 0xFF, all values are reserved for assignment by USB.
6	bDeviceProtocol	1	Protocol	Protocol code (assigned by USB). These codes are qualified by the value of the *bDeviceClass* and the *bDeviceSubClass* fields. If a device supports class-specific protocols on a device basis as opposed to an interface basis, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to zero (0), the device does not use class specific protocols on a device basis. However, it may use class specific protocols on an interface basis. If this field is set to 0xFF (255), the device uses a vendor specific protocol on a device basis.
7	bMaxPacketSize0	1	Number	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid)
8	idVendor	2	ID	Vendor ID (assigned by USB)
10	idProduct	2	ID	Product ID (assigned by manufacturer)
12	bcdDevice	2	BCD	Device release number in Binary-Coded Decimal
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 descriptor describing the device's serial number
17	bNumConfigurations	1	Number	Number of possible configurations

9.7.2 Configuration

The configuration descriptor describes information about a specific device configuration. The descriptor contains a *bConfigurationValue* field with a value that, when used as a parameter to the Set Configuration request, causes the device to assume the described configuration.

The descriptor describes the number of interfaces provided by the configuration. Each interface may operate independently. For example, an ISDN device might be configured with two interfaces, each providing 64 kBs bi-directional channels that have separate data sources or sinks on the host. Another configuration might present the ISDN device as a single interface, bonding the two channels into one 128 kBs bi-directional channel.

When the host requests the configuration descriptor, all related interface and endpoint descriptors are returned (refer to Section 9.5.2).

A USB device has one or more configuration descriptors. Each configuration has one or more interfaces. Each interface has one or more endpoints. An endpoint is not shared among interfaces within a single configuration unless the endpoint is used by alternate settings of the same interface. Endpoints may be shared among interfaces that are part of different configurations without this restriction.

Once configured, devices may support limited adjustments to the configuration. If a particular interface has alternate settings, an alternate may be selected after configuration. Within an interface, an isochronous endpoint's maximum packet size may also be adjusted.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	CONFIGURATION
2	wTotalLength	2	Number	Total length of data returned for this configuration. Includes 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
5	bConfigurationValue	1	Number	Value to use as an argument to Set Configuration to select this configuration
6	iConfiguration	1	Index	Index of string descriptor describing this configuration
7	bmAttributes	1	Bitmap	Configuration characteristics D7 Bus Powered D6 Self Powered D5 Remote Wakeup D4..0 Reserved (reset to 0) A device configuration that uses power from the bus and a local source sets both D7 and D6. The actual power source at runtime may be determined using the Get Status device request. If a device configuration supports remote wakeup, D5 is set to one (1).

Offset	Field	Size	Value	Description
8	MaxPower	1	mA	Maximum power consumption of USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2 mA units (i.e., 50 = 100 mA). Note: A device configuration reports whether the configuration is bus-powered or self-powered. Device status reports whether the device is currently self-powered. If a device is disconnected from its external power source, it updates device status to indicate the device is no longer self-powered. A device may not increase its power draw from the bus when it loses its external power source beyond the amount reported by its configuration. If a device can continue to operate when disconnected from its external power source, it continues to do so. If the device cannot continue to operate, the device fails operations it can no longer support. Host software may determine the cause of the failure by checking the status and noting the loss of the device's power source.

9.7.3 Interface

This descriptor describes a specific interface provided by the associated configuration. A configuration provides one or more interfaces, each with its own endpoint descriptors describing a unique set of endpoints within the configuration. When a configuration supports more than one interface, the endpoints for a particular interface immediately follow the interface descriptor in the data returned by the Get Configuration request. An interface descriptor is always returned as part of a configuration descriptor. It cannot be directly accessed with a Get or Set Descriptor request.

An interface may include alternate settings that allow the endpoints and/or their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. The Set Interface request is used to select an alternate setting or to return to the default setting. The Get Interface request returns the selected alternate setting.

Alternate settings allow a portion of the device configuration to be varied while other interfaces remain in operation. If a configuration has alternate settings for one or more of its interfaces, a separate interface descriptor and its associated endpoints are included for each setting.

If a device configuration supported a single interface with two alternate settings, the configuration descriptor would be followed by an interface descriptor with the *bInterfaceNumber* and *bAlternateSetting* fields set to zero and then the endpoint descriptors for that setting, followed by another interface descriptor and its associated endpoint descriptors. The second interface descriptor's *bInterfaceNumber* field would also be set to zero, but the *bAlternateSetting* field of the second interface descriptor would be set to one.

If an interface only uses endpoint zero, no endpoint descriptors follow the interface descriptor and the interface identifies a request interface that uses the default pipe attached to endpoint zero. In this case, the bNumEndpoints field shall be set to zero.

An interface descriptor never includes endpoint zero in the number of endpoints.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	INTERFACE Descriptor Type
2	bInterfaceNumber	1	Number	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	bAlternateSetting	1	Number	Value used to select alternate setting for the interface identified in the prior field
4	bNumEndpoints	1	Number	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses endpoint zero.
5	bInterfaceClass	1	Class	Class code (assigned by USB) If this field is reset to zero (0), the interface does not belong to any USB specified device class. If this field is set to 0xFF, the interface class is vendor specific. All other values are reserved for assignment by USB.
6	bInterfaceSubClass	1	SubClass	Subclass code (assigned by USB). These codes are qualified by the value of the *bInterfaceClass* field. If the *bInterfaceClass* field is reset to zero (0), this field must also be reset to zero (0). If the *bInterfaceClass* field is not set to 0xFF, all values are reserved for assignment by USB.

Offset	Field	Size	Value	Description
7	bInterfaceProtocol	1	Protocol	Protocol code (assigned by USB). These codes are qualified by the value of the *bInterfaceClass* and the *bInterfaceSubClass* fields. If an interface supports class-specific requests, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to zero (0), the device does not use a class specific protocol on this interface. If this field is set to 0xFF (255), the device uses a vendor specific protocol for this interface.
8	iInterface	1	Index	Index of string descriptor describing this interface

9.7.4 Endpoint

Each endpoint used for an interface has its own descriptor. This descriptor contains the information required by the host to determine the bandwidth requirements of each endpoint. An endpoint descriptor is always returned as part of a configuration descriptor. It cannot be directly accessed with a Get or Set Descriptor request. There is never an endpoint descriptor for endpoint zero.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	ENDPOINT Descriptor Type
2	bEndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0..3: the endpoint number Bit 4..6: reserved, reset to zero Bit 7: direction, ignored for Control endpoints 0 OUT endpoint 1 IN endpoint

Offset	Field	Size	Value	Description
3	bmAttributes	1	Bit Map	This field describes the endpoint's attributes when it is configured using the bConfigurationValue. Bit 0 .. 1:Transfer Type 00 Control 01 Isochronous 10 Bulk 11 Interrupt All other bits are reserved
4	wMaxPacketSize	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. For isochronous endpoints, this value is used to reserve the bus time in the schedule, required for the per frame data payloads. The pipe may, on an ongoing basis, actually use less bandwidth than that reserved. The device reports, if necessary, the actual bandwidth used via its normal, non-USB defined mechanisms. For interrupt, bulk and control endpoints smaller data payloads may be sent, but will terminate the transfer and may or may not require intervention to restart. See Chapter 5.
6	bInterval	1	Number	Interval for polling endpoint for data transfers. Expressed in milliseconds. This field is ignored for Bulk and Control endpoints. For isochronous endpoints this field must be set to one (1). For interrupt endpoints this field may range from 1 to 255.

9.7.5 String

String descriptors are optional. As noted previously, if a device does not support string descriptors, all references to string descriptors within device, configuration, and interface descriptors must be reset to zero.

String descriptors use UNICODE encodings as defined by *The Unicode Standard, Worldwide Character Encoding, Version 1.0, Volumes 1 and 2,* The Unicode Consortium, Addison-Wesley Publishing Company, Reading, Massachusetts. The strings in a USB device may support multiple languages. When requesting a string descriptor, the requester specifies the desired language using a sixteen-bit language ID (LANGID) defined by Microsoft for Windows as described by *Developing International Software for Windows 95 and Windows NT,* Nadine Kano, Microsoft Press, Redmond, Washington. String index zero (0) for all languages returns an array of two-byte LANGID codes supported by the device. A USB device may omit all string descriptors.

The UNICODE string descriptor is not NULL terminated. The string length is computed by subtracting two from the value of the first byte of the descriptor.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	STRING Descriptor Type
2	bString	N	Number	UNICODE encoded string

9.8 Device Class Definitions

All devices must support the above registers and descriptor definitions. Most devices provide additional registers and possibly, descriptors for device-specific extensions. In addition, devices may provide extended services which are common to a group of devices. In order to define a class of devices, the following information must be provided to completely define the appearance and behavior of the device class.

9.8.1 Descriptors

If the class requires any specific definition of the standard descriptors, the class definition must include those requirements as part of the class definition. In addition, if the class defines a standard extended set of descriptors, they must also be fully defined in the class definition. Any extended descriptor definitions should follow the approach used for standard descriptors; for example, all descriptors should begin with a length field.

9.8.2 Interface(s) and Endpoint Usage

When a class of devices is standardized, the interfaces used by the devices including how endpoints are used must be included in the device class definition. Devices may further extend a class definition with proprietary features as long as they meet the base definition of the class.

9.8.3 Requests

All of the requests specific to the class must be defined.

9.9 Device Communications

The USB communications model characterizes data and control traffic between the host and a given device across the USB interconnect. The host and the device are divided into the distinct layers shown in Figure 9-2.

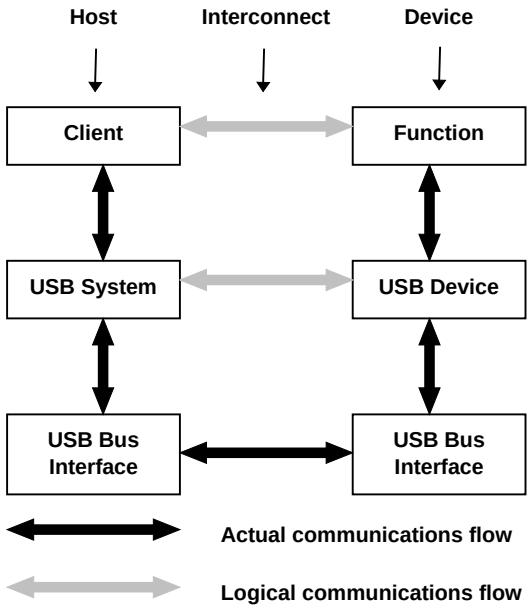

Figure 9-2. Interlayer Communications Model

The actual communication on the host, as indicated by vertical arrows, takes place via SPIs. The interlayer relationships on the device are implementation-specific. Between the host and device, all communications must ultimately occur on the physical USB wire. However, there are logical host-device interfaces between horizontal layers. Between client software, resident on the host, and the function provided by the device, the communications are typified by a contract based on the needs of the application currently using the device and the capabilities provided by the device. This client-function interaction creates the requirements for all of the underlying layers and their interfaces.

This section describes the communications model from the point of view of the device and its layers. Figure 9-3 illustrates, based on the overall view introduced in Chapter 8, the device's view of its communication with the host.

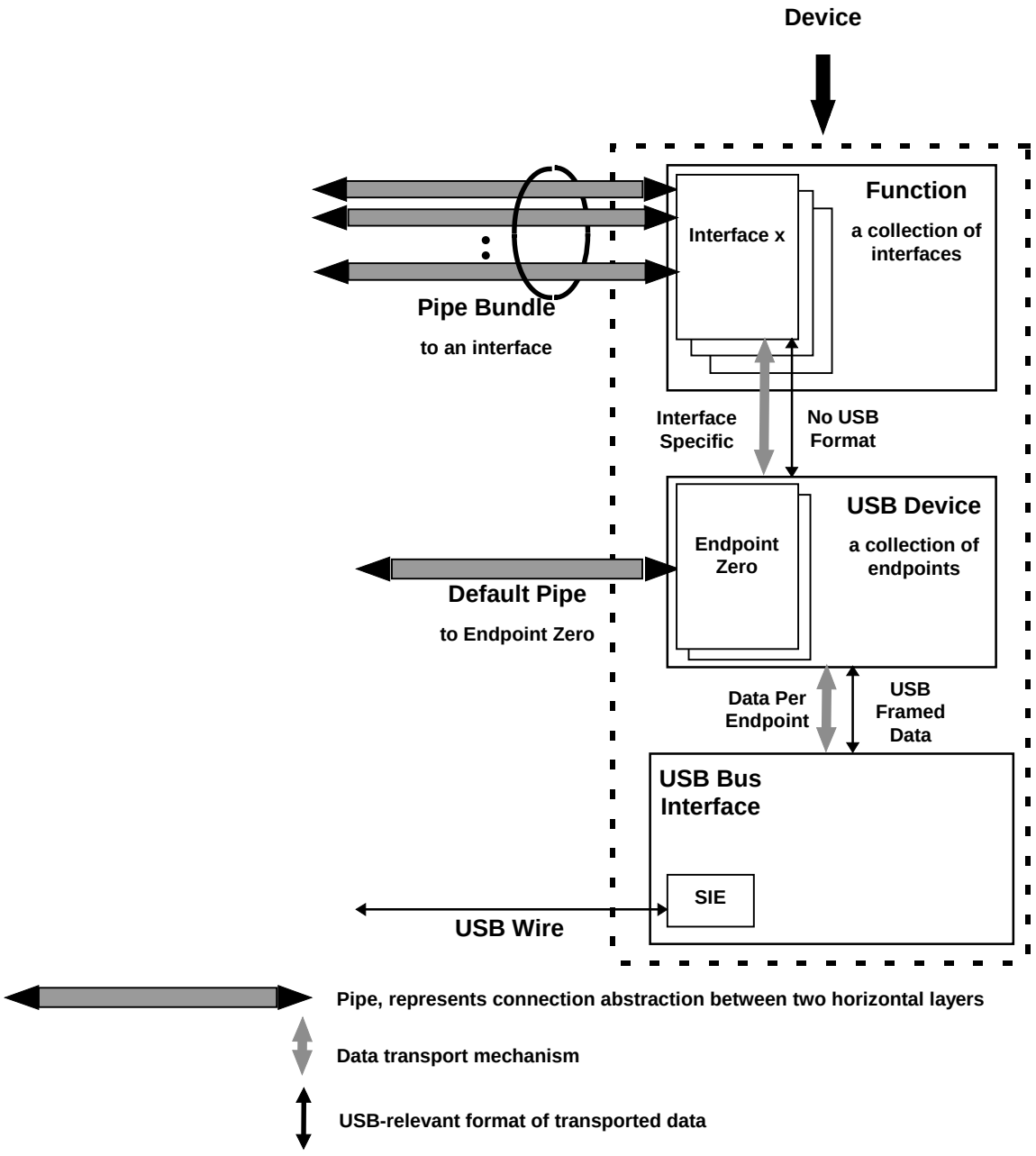

Figure 9-3. Device Communications

The USB Bus Interface handles interactions among the electrical and protocol layers (refer to Chapters 7 and 8). The USB Device layer presents a uniform abstraction of the USB device to the host. It is this layer that is primarily described here. The Function layer uses the capabilities provided by the USB Device layer, combined into a given interface, to support the requirements of a host-based application.

A USB Device acts as a collection of endpoints, each capable of supporting different types of pipes. Each pipe can support one of the following transfer types at a time:

- Control
- Isochronous
- Interrupt
- Bulk

These transfer types are described in more detail in Chapter 5. Each of the transfer types, however, requires the associated endpoint to behave in a certain fashion. A given endpoint may support a variety of transfer types. However, once a pipe is associated with an endpoint, the endpoint only uses a single transfer type. In this section, when discussing the behavior of an endpoint for a given transfer type, it is assumed that the endpoint has been associated with a pipe supporting that specific transfer type. The basic communication mechanisms used by endpoints are:

- Pipe Mode
- Start of Frame (SOF) Synchronization
- Handshakes
- Data Toggles

The mode of a pipe indicates whether the data flow across the pipe is stream or message mode. Devices may use the SOF as generated by the host to synchronize their internal clocks. Devices may use handshakes and data toggles to implement error and flow control.

Traffic between a client and a function may require a certain transport rate. The client, USB and the function all will be using, at best, slightly different clock rates. To ensure that all of the required data can be delivered with minimum buffering required, the various clocks must be synchronized. Refer to Chapter 5 for a discussion of the synchronization options. Additionally, in order to support the just-in-time delivery capability implied by clock synchronization, the size of the data packets transmitted between the host and the device will be normalized such that variations in size over time are minimized. To support data flows in which the loss of data is acceptable as long as the loss can be accurately communicated, sample headers may be used by the host and the device to communicate the expected transmission volumes. Refer to Chapter 5 for the definition of sample headers.

These basic communication mechanisms are described, from the device's point of view, in greater detail below. Each of the different transfer types uses these basic communication mechanisms in different ways.

9.9.1 Basic Communication Mechanisms

This section describes in more detail the basic communication mechanisms as supported by the USB Device layer. Table 9-5 summarizes the USB communication mechanisms.

9.9.1.1 Pipe Mode

A pipe supports either stream or message mode transfers. In stream mode, the data flow is considered to be a unidirectional serial stream of samples. In message mode, data is delivered as a related set of bytes. Message mode pipes are always considered to be capable of being bi-directional.

A stream mode pipe is always unidirectional. When in the stream mode, the endpoint expects to receive a token either requesting the endpoint to send data or alerting it that data will be sent to it. The amount of data sent will always be equal to or less than the current MaxPacketSize for the endpoint.

Message transfers begin with a command from the host to the device. The device may respond to the command with data, the host may follow the command with data for the device, or the command may

require no data to be transmitted in which case a NULL data packet will be sent. In message mode, an endpoint must keep track of where it is in the phase sequence defined by the mode. An endpoint expects the first transaction of a message sequence to be a setup for the subsequent communication. After setup is received, an endpoint usually expects to receive a token requesting the endpoint to send data (IN token) or alerting it that data will be sent to the endpoint (OUT Token). The endpoint will know what the direction of the subsequent transactions will be, based on the setup command that started the series of transactions. Some setup commands do not require subsequent transactions to or from an endpoint. Setup transactions are always eight bytes or less. The subsequent transactions will always be of a size equal to or less than the current MaxPacketSize for the endpoint.

9.9.1.2 Synchronization

The host provides a special SOF token to the bus at regularly timed intervals. The interval between SOF's, within error tolerances (refer to Chapter 7), is 1 ms. Endpoints may use the receipt of this token to synchronize their associated clock to the USB clock. This enables endpoints to match their rate of data consumption or production to the host's rate.

Not all endpoints require SOF synchronization. Some endpoints requiring synchronization have clocks which cannot be synchronized to the 1 ms bus clock provided by USB. Such devices have two choices. They can attempt to have the entire USB synchronize to them or such devices may periodically adjust their transfer rate as they compensate for the difference between the USB clock and their own clock.

USB provides for a maximum of one client per USB instance to adjust the host's SOF generation. This client performs the adjustment based on feedback provided by an associated device. The rate of SOF token generation remains 1 ms, however. Refer to Chapter 10 for a complete discussion of this adjustment mechanism. If an endpoint has not been configured to adjust the USB clock using the SOF handshake, or if the endpoint is not capable of so adjusting the clock, then the endpoint must continually adjust its data flow.

Therefore, as noted above, there are three possible types of synchronization interaction for an endpoint with regard to SOF. The endpoint may:

1. Synchronize its clock exactly to the existing USB clock.
2. Adjust the bus clock.
3. Synchronize with the host by adjusting its data flow.

It is important to note that an endpoint requiring synchronization, which cannot implement the type of synchronization described in (1), and which can implement the type described in (2), must also implement the type described in (3). This is because such an endpoint cannot be guaranteed that it will be chosen as the endpoint to adjust the bus clock. Only one device on the entire USB will be used to adjust the SOF.

9.9.1.3 Handshakes

Endpoints use the handshake phase of USB transactions to communicate error and data flow needs to the host. Endpoints may also receive handshakes from the host to communicate error conditions. The types of handshakes used by the endpoint vary according to the transfer type supported. These handshakes are described in detail in Chapter 8.

9.9.1.4 Data Toggles

When an error or flow control situation occurs, some pipes are allowed to skip the frame in which the condition occurred and transfer the data scheduled for that frame during a subsequent frame. In some cases, it is possible that the receiver of the data had indicated to the transmitter that the data had been successfully received, but that the transmitter believes, due to a bus error, that the data was not received successfully. The transmitter will then retransmit the same data. The receiver needs some mechanism to understand that the data it is now receiving is a retransmission of data it has already received and not new data.

USB provides this information by using data toggles which are the PIDs for the data phase of transactions. Depending on the transfer type, the endpoint needs to understand data toggles and generate or process data PIDs accordingly. Refer to Chapter 8 for a more complete discussion of data toggles.

Table 9-5. USB Communications Mechanisms

	Control	Isochronous	Interrupt	Bulk
Pipe Mode	Message	Stream	Stream	Stream
Synchronization	None	Bus, external, or software	None	None
Handshakes	Yes	Not used	Yes	Yes
Data Toggles	Yes	Ignored	Yes	Yes
Required Buffering	Minimum of Eight Bytes	Twice frame traffic	Single transaction	Single transaction
Error and Status Handling	Guaranteed delivery reports fatal errors only	Reports missing or corrupt data - no retries	Guaranteed delivery reports fatal errors only	Guaranteed delivery reports fatal errors only