Chapter 10
USB Host: Hardware and Software

10.1 Overview of the USB Host

The USB interconnect supports data traffic between a host and a USB device. This chapter describes the host interfaces necessary to facilitate USB communication between a software client, resident on the host, and a function implemented on a device.

The basic flow and interrelationships of the USB communications model are shown in Figure 10-1.

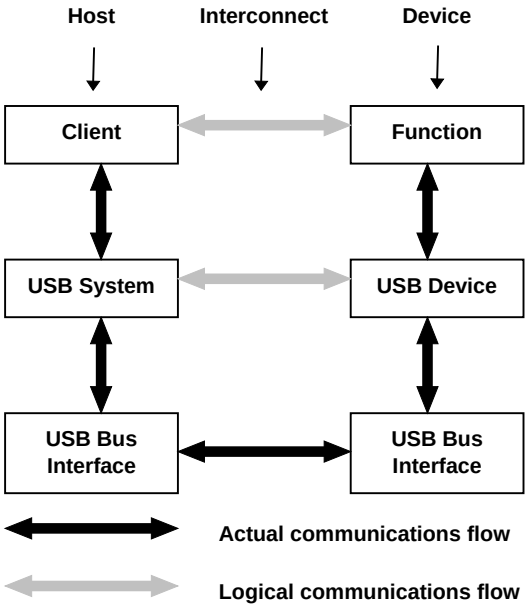

Figure 10-1. Interlayer Communications Model

The host and the device are divided into the distinct layers depicted above. The actual communication on the host is indicated by vertical arrows. The corresponding interfaces on the device are implementation-specific. All communications between the host and device ultimately occur on the physical USB wire. However, there are logical host-device interfaces between each horizontal layer. These communications, between client software resident on the host and the function provided by the device, 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 chapter describes this model from the point-of-view of the host and its layers. Figure 10-2 describes, based on the overall view introduced in Chapter 5, the host's view of its communication with the device.

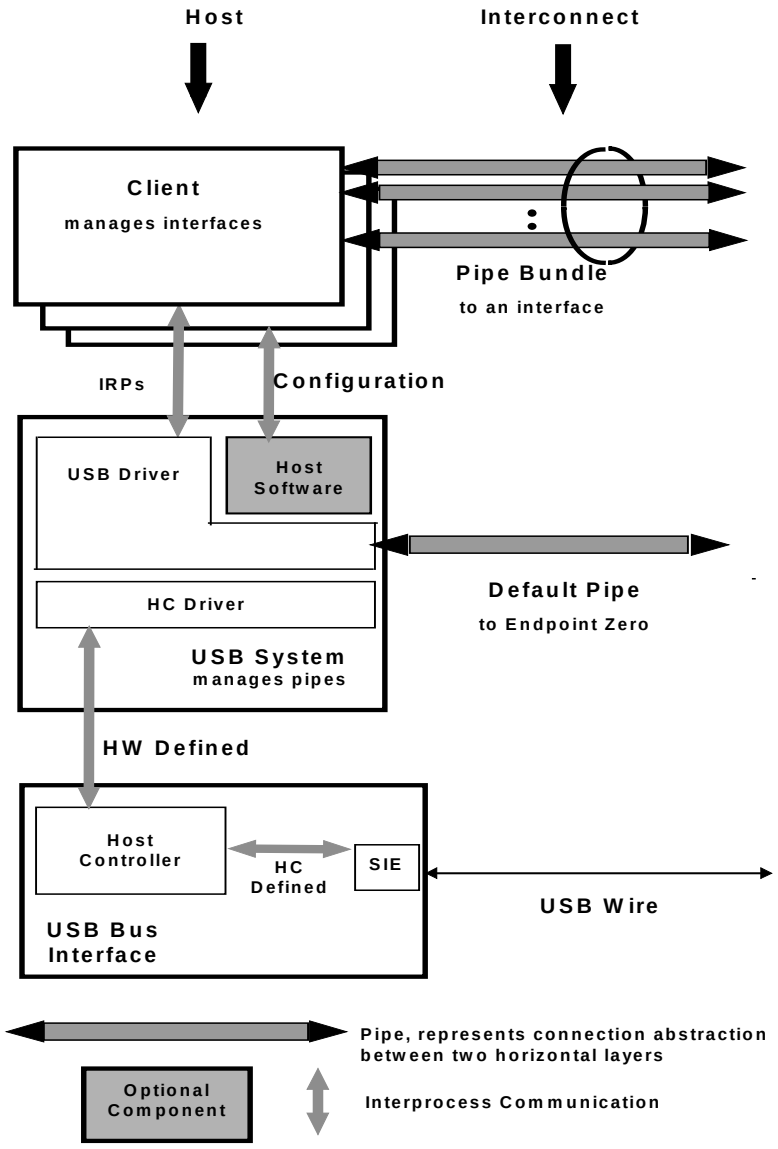

Figure 10-2. Host Communications

There is only one host for a given USB. The major layers of a host are:

- USB Bus Interface
- USB System
- Client

The USB Bus Interface handles interactions for the electrical and protocol layers (refer to Chapters 7 and 8). From the interconnect point-of-view, a similar USB Bus Interface is provided by both the USB device and the host, as exemplified by the Serial Interface Engine (SIE). On the host, however, the USB Bus Interface has additional responsibilities due to the unique role of the host on the USB and is implemented as the host controller. The host controller also has an integrated, or root, hub providing attachment points to the host.

The USB System uses the host controller to manage data transfers between the host and the USB device. The interface between the USB System and the host controller is dependent on the hardware definition of

the host controller. The USB System, in concert with the host controller, performs the translation between the client's view of data transfers and the USB transactions appearing on the interconnect. This includes the addition of any USB feature support such as protocol wrappers. The USB System is also responsible for managing USB resources; e.g., bandwidth, such that client access to USB is possible.

The USB System has three basic components:

- Host Controller Driver
- USB Driver
- Host Software

The Host Controller Driver (HCD) exists to more easily map the various possible host controller implementations into the USB System, such that a client can interact with its device without knowing to which host controller the device is connected. The USB Driver (USBD) provides the basic host interface (USBDI) for clients to USB devices. The interface between HCD and USBD is known as the Host Controller Driver Interface (HCDI). This interface is never available directly to clients and thus is not defined by the USB specification. A particular HCDI is, however, defined by each operating system that supports various host controller implementations.

USBD provides data transfer mechanisms in the form of I/O Request Packets (IRPs), which consist of a request to transport data across a specific pipe. In addition to providing data transfer mechanisms, USBD is responsible for presenting to its clients an abstraction of a USB device which can be manipulated for configuration and state management. As part of this abstraction, USBD owns the default pipe (see Chapters 5 and 9) through which all USB devices are accessed for the purposes of standard USB control. This default pipe represents a logical communication between USBD and the abstraction of a USB device as shown in Figure 10-2.

In some operating systems, additional non-USB host software is available which provides configuration and loading mechanisms to device drivers. In such operating systems, the device driver shall use the provided interfaces instead of directly accessing the USBDI mechanisms.

The client layer describes all of those software entities which are responsible for directly interacting with their peripherals. When each device is independently attached to the system, these clients might interact directly with the peripheral hardware. The shared characteristics of USB place a USB software stack between the client and its device; that is, a client cannot directly access the device's hardware.

Overall, the host layers provide the following capabilities:

- Detecting the attachment and removal of USB devices
- Managing USB standard control flow between the host and USB devices
- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Controlling the electrical interface between the host controller and USB devices, including the provision of a limited amount of power

The following sections describe these responsibilities and the requirements placed on the USBDI in greater detail. The actual interfaces used for a specific combination of host platform and operating system are described in the appropriate Operating System Environment Guide.

All hubs provide a status change pipe (see Chapter 11) on which status changes for hubs and their ports are reported. This includes a notification of when a USB device is attached to or removed from one of their ports. A USBD client generically known as the hub driver receives these notifications as owner of the hub's status change pipe. For device attachments, the hub driver then initiates the device configuration process. In some systems, this hub driver is a part of the host software provided by the operating system for managing devices.

10.1.2 Control Mechanisms

Control information may be passed between the host and a USB device using in-band or out-of-band signaling. In-band signaling mixes control information with data in a pipe outside the awareness of the host. Out-of-band signaling places control information in a separate pipe.

There is a message pipe called the default pipe for each attached USB device. This logical association between a host and a USB device is used for USB standard control flow such as device enumeration and configuration. The default pipe provides a standard interface to all USB devices. The default pipe may also be used for device-specific communications, as mediated by USBD which owns the default pipes of all of the USB devices.

A particular USB device may allow the use of additional message pipes to transfer device-specific control information. These pipes use the same communications protocol as the default pipe, but the information transferred is specific to the USB device and is not standardized by the Universal Serial Bus Specification.

USBD supports the sharing of the default pipe, which it owns and uses, with its clients. It also provides access to any other control pipes associated with the device.

10.1.3 Data Flow

The host controller is responsible for transferring streams of data between the host and USB devices. These data transfers are treated as a continuous stream of bytes. USB supports four basic types of data transfers:

- Control transfers
- Isochronous transfers
- Interrupt transfers
- Bulk transfers

For additional information on transfer types, refer to Chapter 5.

Each device presents one or more interfaces which a client may use to communicate with it. Each interface is composed of zero or more pipes which individually transfer data between the client and a particular endpoint on the device. USBD establishes interfaces and pipes at the explicit request of host software. The host controller provides service based on parameters provided by the host software when the configuration request is made.

A pipe has several characteristics based on the delivery requirements of the data to be transferred. Examples of these characteristics are: the rate at which data needs to be transferred, whether data is provided at a steady rate or sporadically, how long data may be delayed before delivery, and whether the loss of data being transferred is catastrophic.

A USB device endpoint describes the characteristics required for a specific pipe. Endpoints are described as part of a USB device's characterization information. For additional details, refer to Chapter 9.

10.1.4 Collecting Status and Activity Statistics

As a common communicant for all control and data transfers between the host and USB devices, the USB System and the host controller is well positioned to track status and activity information. Such information is provided upon request to host software allowing that software to manage status and activity information.

10.1.5 Electrical Interface Considerations

The host provides power to USB devices attached to the root hub. The amount of power provided by a port is specified in the discussion of hubs in Chapter 11.

10.2 Host Controller Requirements

In all implementations, host controllers perform the same basic duties with regard to the USB and its attached devices. These basic duties are described below.

The host controller has requirements from both the host and the USB. The following is a brief overview of the functionality provided. Each capability is discussed in detail in subsequent subsections.

State Handling	As a component of the host, the host controller reports and manages its states.
Serializer/deserializer	For data transmitted from the host, the host controller converts protocol and data information from its native format to a bit stream transmitted on USB. For data being received into the host, the reverse operation is performed.
Frame Generation	The host controller produces SOF tokens at a period of 1 ms.
Data Processing	The host controller processes requests for data transmission to and from the host.
Protocol Engine	The host controller supports the protocol specified by USB.
Transmission Error Handling	All host controllers exhibit the same behavior when detecting and reacting to the defined error categories.

The following subsections present a more detailed discussion of the required capabilities of the host controller.

10.2.1 State Handling

As a normal component in the host, the host controller has a series of states which the USB System manages. Additionally, the host controller has two areas of USB-relevant state:

- Root Hub
- State change propagation

The root hub presents to the hub driver the same standard states as other USB devices. The host controller supports these states and their transitions for the hub. For detailed discussions of USB states, including their interrelations and transitions, refer to Chapter 9.

The overall state of the host controller is inextricably linked with that of the root hub and of the overall USB. Any host controller state changes which are visible to attached devices must be reflected in corresponding device state changes such that the resulting host controller and device states are consistent.

USB devices request a wakeup through the use of resume signaling (refer to Chapter 11), which causes hubs to tear down connectivity, and devices to return to their configured state. The host controller itself may cause a resume event through the same signaling method. The host controller must notify the rest of the host of a resume event through a mechanism or mechanisms specific to that system's implementation.

10.2.2 Serializer/Deserializer

The actual transmission of data across the physical USB takes places as a serial bit stream. A Serial Interface Engine (SIE), whether implemented as part of the host or a USB device, handles the serialization and deserialization of USB transmissions. On the host, this SIE is part of the host controller.

10.2.3 Frame Generation

It is the host controller's responsibility to partition USB time into 1 ms quantities called "frames". Frames are created by the host controller through issuing Start of Frame (SOF) tokens at 1.00 ms intervals. The SOF token is the first transmission in the frame period. After issuing a SOF token, the host controller is free to transmit other transactions for the remainder of the frame period. When the host controller is in its normal operating state, SOF tokens must be continuously generated at the 1 ms periodic rate, regardless of the other bus activity or lack thereof. If the host controller enters a state where it is not providing power on the bus, it must not generate SOFs. Also, if the host controller is not generating SOFs, it may enter a power-reduced state.

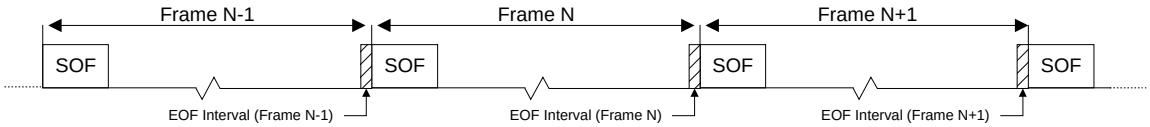

Figure 10-3. Frame Creation

The SOF token holds the highest priority access to the bus. Babble circuitry in hubs electrically isolates any active transmitters during the End of Frame (EOF) interval, providing an idle bus for the SOF transmission.

- The host controller must allow the length of the USB frame to be adjusted by +/- 1 bit time (refer to Section 10.5.3.2.2). The host controller maintains the current frame number which may be read by the USB system. The current frame number is used to uniquely identify one frame from another.
- Incremented at the end of every frame period.
- Valid through the subsequent frame.

The host transmits the lower 11-bits of the current frame number in each SOF token transmission. When requested from the host controller, the current frame number is the frame number in existence at the time the request was fulfilled. The current frame number as returned by the host (host controller or HCD) is at least 32 bits, although the host controller itself is not required to maintain more than 11 bits.

The host controller shall cease transmission during the EOF interval. When the EOF interval begins, any transactions scheduled specifically for the frame which has just passed are retired. If the host controller is executing a transaction at the time the EOF interval is encountered, the host controller terminates the transaction.

10.2.4 Data Processing

The host controller is responsible for receiving data from the USB System and sending it to the USB and for receiving data from the USB and sending it to the USB System. The particular format used for the data communications between the USB System and the host controller is implementation specific, within the rules for transfer behavior described in Chapter 5.

10.2.5 Protocol Engine

The host controller manages the USB protocol level interface. It inserts the appropriate protocol information for outgoing transmissions. It also strips and interprets, as appropriate, the incoming protocol information.

10.2.6 Transmission Error Handling

The host controller must be capable of detecting the following transmission error conditions, which are defined from the host's point-of-view:

- Time-out conditions after a host-transmitted token or packet. These errors occur when the addressed endpoint is unresponsive or when the structure of the transmission is so badly damaged that the targeted endpoint does not recognize it.
- Data errors resulting in missing or invalid transmissions.
  - The host controller sends or receives a packet shorter than that required for the transmission; for example, a transmission extending beyond EOF or a lack of resources available to the host controller.
  - An invalid CRC field on a received data packet.
- Protocol errors.
  - An invalid handshake PID. For example, a malformed or an inappropriate handshake.
  - A false EOP.
  - A bit stuffing error.

For each bulk, command, and interrupt transaction, the host must maintain an error count tally. Errors result from the types of conditions described above, not as a result of an endpoint NAKing a request. This value reflects the number of times the transaction has encountered a transmission error. If the error count tally for a given transaction reaches three, the host retires the transfer. When a transfer is retired due to excessive errors, the last error type will be indicated. Isochronous transactions are attempted only once, regardless of outcome, and, therefore, no error count is maintained for this type.

10.3 Overview of Software Mechanisms

HCD and USBD present software interfaces based on different levels of abstraction. They are, however, expected to operate together in a specified manner to satisfy the overall requirements of the USB system (see Figure 10-2). The requirements for the USB software stack are expressed primarily as requirements on USBDI. The division of duties between USBD and HCD is not defined. However, the one requirement of HCDI which must be met is that it support, in the specified operating system context, multiple definitions of host controllers.

HCD provides an abstraction of the host controller and an abstraction of the host controller's view of data transfer across USB. USBD provides an abstraction of the USB device and of the data transfers between the client of USBD and the function on the USB device. Overall, the USB software stack acts as a facilitator for transmitting data between the client and the function and as a control point for the USB-specific interfaces of the USB device. As part of facilitating data transfer, the USB software provides buffer management capabilities and allows the synchronization of the data transmittal to the needs of the client and the function.

The specific requirements on USBDI are described later in this chapter. The exact functions which fulfill these requirements are described in the relevant Operating System Environment Guide for HCDI and USBDI. The procedures involved in accomplishing data transfers via USBDI are described below.

10.3.1 Device Configuration

Different operating system environments perform device configuration using different software components and different sequences of events. A specific operating system method is not assumed by the USB System. However, there are some basic requirements which must be fulfilled by any USB System implementation. In some operating systems, these requirements are met by existing host software. In others, the USB System provides the capabilities.

The USB System assumes a specialized client of USBD, called a hub driver, which acts as a clearing house for addition of devices to and removal of devices from a particular hub. Once the hub driver receives such notifications, it will employ additional host software and other USBD clients, in an operating system specific manner, to recognize and configure the device. This model is the basis of the discussion below.

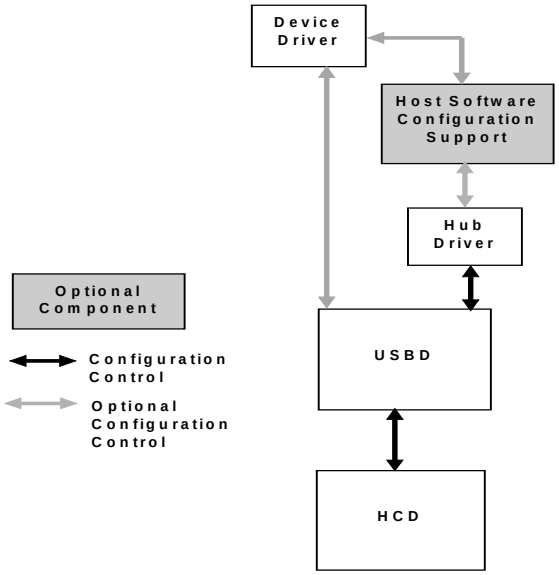

Figure 10-4. Configuration Interactions

When a device is attached, the hub driver receives a notification from the hub detecting the change. The hub driver, using the information provided by the hub, requests a device identifier from USBD. USBD in turn sets up the default pipe for that device and returns a device identifier to the hub driver.

The device is now ready to be configured for use. For each device, there are three types of configuration which must be complete before that device is ready for use:

1. Device configuration. This includes setting up all of the device's USB parameters and allocating all USB host resources which are visible to the device. This is accomplished by setting the configuration value on the device. A limited set of configuration changes, such as alternate setting, are allowed without totally reconfiguring the device. Once the device is configured, it is, from its point of view, ready for use.
2. USB configuration. In order to actually create a USBD pipe ready for use by a client, additional USB information, not visible to the device, must be specified by the client. This information, known as the **POLICY** for the pipe, describes how the client will use the pipe. This includes such items as the maximum amount of data the client will transfer with one IRP, the maximum service interval the client will use, the client's notification identification, and so on.
3. Function configuration. Once 1 and 2 above have been accomplished, the pipe is now completely ready for use from the USB point of view. However, additional vendor- or class-specific setup may be required before the pipe can actually be used by the client. This configuration is a private matter between the device and the client and is not standardized by USBD.

The device and USB configuration requirements are described below.

The actual device configuration is performed by the responsible configuring software. Depending on the particular operating system implementation, the software responsible for configuration can include:

- The hub driver
- Other host software
- A device driver

The configuring software first reads the device descriptor, then requests the description for each possible configuration. It may use the information provided to load a particular client, such as a device driver, which initially interacts with the device. The configuring software, perhaps with input from that device driver, chooses a configuration for the device. Setting the device configuration sets up all of the endpoints

on the device and returns a collection of interfaces to be used for data transfer by USBD clients. Each interface is a collection of pipes owned by a single client.

This initial configuration uses the default settings for interfaces and the default bandwidth for each endpoint. A USBD implementation may additionally allow the client to specify alternate interfaces when selecting the initial configuration. The USB System will verify that the resources required for the support of the endpoint are available and, if so, will allocate the bandwidth required. Refer to Section 10.3.2 for a discussion of resource management.

The device is now configured, but the created pipes are not yet ready for use. The USB configuration is accomplished when the client initializes each pipe by setting a policy to specify how it will interact with the pipe. Among the information specified is the client's maximum service interval and notification information. Among the actions taken by the USB System, as a result of setting the policy, is determining the amount of buffer working space required beyond the data buffer space provided by the client. The size of the buffers required is based upon the usage chosen by the client and upon the per transfer needs of the USB System.

The client receives notifications when IRPs complete, successfully or due to errors. The client may also wake up independently of USB notification to check the status of pending IRPs.

The client may also choose to make configuration modifications such as enabling an alternate setting for an interface or changing the bandwidth allocated to a particular pipe. In order to perform these changes, the interface or pipe, respectively, must be idle.

10.3.2 Resource Management

Whenever a pipe is setup by USBD for a given endpoint, the USB System must determine if it can support the pipe. The USB System makes this determination based on the requirements stated in the endpoint descriptor. One of the endpoint requirements which must be supported in order to create a pipe for an endpoint is the bandwidth necessary for that endpoint's transfers. There are two stages to check for available bandwidth. First the maximum execution time for a transaction is calculated. Then, the frame schedule is consulted to determine if the indicated transaction will fit.

The allocation of the guaranteed bandwidth for isochronous and interrupt pipes, and the determination of whether a particular control or bulk transaction will fit into a given frame, can be determined by a software heuristic in the USB System. If the actual transaction execution time in the host controller exceeds the heuristically determined value, the host controller is responsible for ensuring that frame integrity is maintained (refer to Section 10.2.3). The following discussion describes the requirements on the USB System heuristic.

In order to determine if bandwidth can be allocated, or if a transaction can be fit into a particular frame, the maximum transaction execution time must be calculated. The calculation of the maximum transaction execution time requires that the following information be provided: (note that some of this information may be provided by an agent other than the client)

- Number of data bytes (MaxPacketSize) to be transmitted.
- Transfer type.
- Depth in the topology. If lesser precision is allowed, the maximum topology depth may be assumed.

This calculation must include the bit transmission time, the signal propagation delay through the topology, and any implementation specific delays such as preparation or recovery time required by the host controller itself. Refer to Chapter 5 for examples of formulas that can be used for such calculations.

10.3.3 Data Transfers

The basis for all client-function communication is the interface: a bundle of related pipes associated with a particular USB device.

A given interface is managed by exactly one client on the host. The client initializes each pipe of an interface by setting the policy for that pipe. This includes the maximum amount of data to be transmitted per IRP and the maximum service interval for the pipe. A service interval is stated in milliseconds and describes the interval over which an IRP's data will be transmitted for an isochronous pipe. It describes the polling interval for an interrupt pipe. The client is notified when a specified request is completed. The client manages the size of each IRP such that its duty cycle and latency constraints are maintained. Additional policy information includes the notification information for the client.

The client provides the buffer space required to hold the transmitted data. The USB System uses the policy to determine the additional working space it will require .

The client views its data as a contiguous serial stream, which it manages in a similar manner to those streams provided over other types of bus technologies. Internally, the USB System may, depending on its own policy and any host controller constraints, break the client request down into smaller requests to be sent across the USB. However, two requirements must be met whenever the USB System chooses to undertake such division:

- The division of the data stream into smaller chunks is not visible to the client.
- USB samples are not split across bus transactions. Refer to Chapter 9 for a definition of USB sample and its relationship to the pipe's natural sample size.

When a client wishes to transfer data, it will send an IRP to USBD. Depending on the direction of data transfer, a full or empty data buffer will be provided. When the request is complete (successfully or due to an error condition), the IRP and its status is returned to the client. Where relevant, this status is also provided on a per transaction basis.

10.3.4 Common Data Definitions

In order to allow the client to receive request results as directly as possible from its device, it is desirable to minimize the amount of processing and copying required between the device and the client. To facilitate this, some control aspects of the IRP are standardized such that the information provided by the client may be directly used by different layers in the stack. The particular format for this data is dependent on the actualization of the USBDI in the operating system. Some data elements may in fact not be directly visible to the client at all, but are generated as a result of the client request.

The following data elements define the relevant information for a request:

- Identification of the pipe associated with the request. Identifying this pipe also describes information such as transfer type for this request.
- Notification identification for the particular client.
- Location and length of data buffer which is to be transmitted or received.
- Completion status for the request. Both the summary status, and, as required, detailed per-transaction status must be provided.
- Location and length of working space. This is implementation dependent.

The actual mechanisms used to communicate requests to USBD are operating system specific. However, beyond the requirements stated above for what request-related information must be available, there are also requirements on how requests will be processed. The basic requirements are described in Chapter 5. Additionally, USBD provides a mechanism to designate a group of isochronous IRPs for which the transmission of the first transaction of each IRP will occur in the same frame. USBD also provides a mechanism for designating an uninterruptable set of vendor- or class-specific requests to a default pipe. No other requests to that default pipe, including standard, class, or vendor request may be inserted in the execution flow for such an uninterruptable set. If any request in this set fails, the entire set is retired.

10.4 Host Controller Driver

Host Controller Driver (HCD) is an abstraction of host controller hardware and the host controller's view of data transmission over the USB. The HCDI meets the following requirements:

- Provides an abstraction of the host controller hardware.
- Provides an abstraction for data transfers by the host controller across the USB interconnect.
- Provides an abstraction for the allocation (and de-allocation) of host controller resources, to support guaranteed service to USB Devices.
- Presents the root hub and its behavior according to the hub class definition. This includes supporting the root hub such that the hub driver interacts with the root hub exactly as it would for any hub. In particular, even though a root hub can be implemented in a combination of hardware and software, the root hub responds initially to the default device address (from a client perspective), returns descriptor information, supports having its device address set, and supports the other hub class requests. However, bus transactions may or may not need to be generated to accomplish this behavior given the close integration possible between the host controller and the root hub.

HCD provides a software interface (HCDI) which implements the required abstractions. The function of HCD is to provide an abstraction which hides the details of the host controller hardware. Below the host controller hardware is the physical USB and all the attached USB devices.

HCD is the lowest tier in the USB software stack. HCD has only one client: the Universal Serial Bus Driver (USBD). USBD maps requests from many clients to the appropriate HCD. A given HCD may manage many host controllers.

HCDI is not directly accessible from a client. Therefore, the specific interface requirements for HCDI are not discussed here.

10.5 Universal Serial Bus Driver

A Universal Serial Bus Driver (USBD) provides a collection of mechanisms that operating system components, typically device drivers, use to access USB devices. The only access to USB devices is that provided by USBD. USBD implementations are operating system specific. The mechanisms provided by USBD are implemented using as appropriate and augmenting as necessary the mechanisms provided by the operating system environment in which it runs. The following discussion centers on the basic capabilities required for all USBD implementations. For specifics of the USBD operation within a specific environment, see the relevant Operating System Environment Guide for USBD. A single instance of the USBD directs accesses to one or more Host Controller Drivers (HCDs) that in turn connect to one or more USB host controllers. If allowed, how USBD instancing is managed is dependent upon the operating system environment. However, from the client's point-of-view, the USBD with which it communicates manages all of the attached USB devices.

10.5.1 Overview

Clients of USBD direct commands to devices or move streams of data to or from pipes. USBD presents two groups of software mechanisms to clients: Command mechanisms and Pipe mechanisms.

Command mechanisms allow clients to configure and control USBD operation as well as to configure and generically control a USB device. In particular, Command mechanisms provide all access to the device's default pipe.

Pipe mechanisms allow a USBD client to manage device specific data and control transfers. Pipe mechanisms do not allow a client to directly address the device's default pipe.

Figure 10-5 presents an overview of the USBD structure.

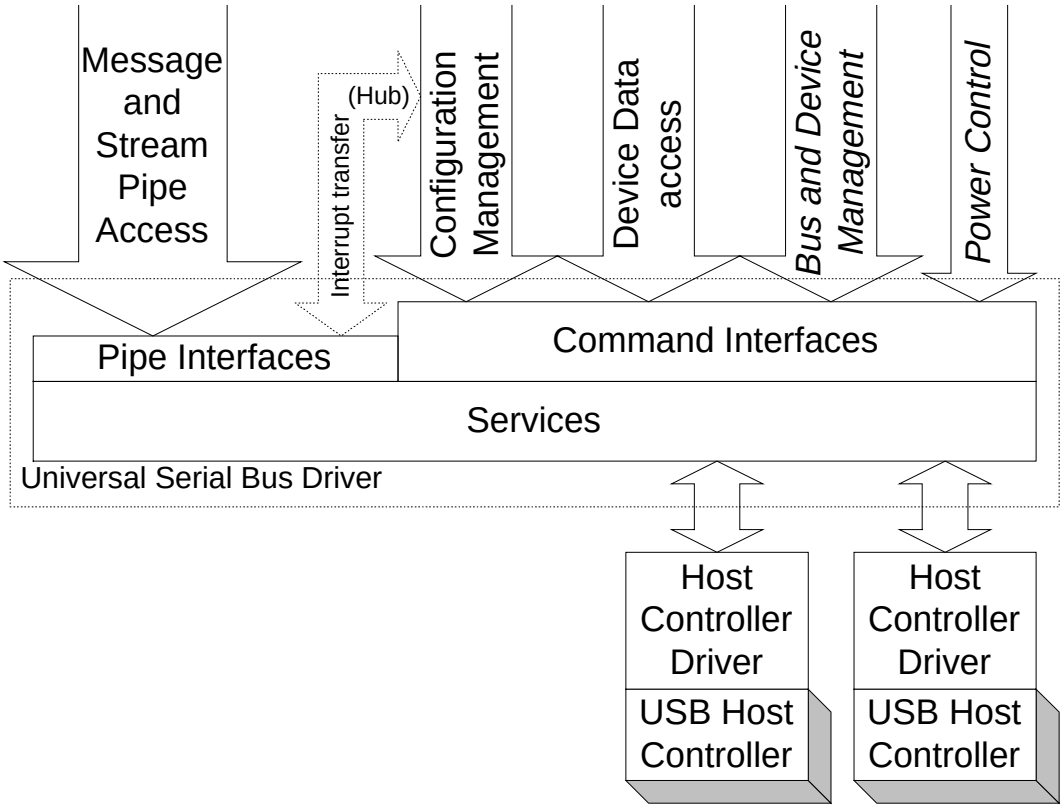

Figure 10-5. Universal Serial Bus Driver Structure

10.5.1.1 Initialization

Specific USBD initialization is operating system dependent. When a particular USB managed by USBD is initialized, the management information for that USB is also created. Part of this management information is the default address device and its default pipe.

Whenever a device is attached to a USB, it responds to a special address known as the default address (refer to Chapter 9) until its unique address is assigned by the hub driver. In order for the USB System to interact with the new device, the default address device for that particular USB and that default address device's default pipe must always be available to the hub driver whenever a device is attached. The default address device and its default pipe are created by the USB System when a new USB is initialized.

10.5.1.2 USBD Pipe Usage

Pipes are the method by which a device endpoint is associated with a host software entity. Pipes are owned by exactly one such entity on the host. Although the basic concept of a pipe is the same no matter who the owner, some distinction of capabilities provided to the USBD client occurs between two groups of pipes:

- Default pipes, which are owned and managed by USBD
- All other pipes, which are owned and managed by clients of USBD

Default pipes are never directly accessed by clients, although they are often used to fulfill some part of client requests relayed via Command mechanisms.

10.5.1.2.1 Default Pipes

USBD is responsible for allocating and managing appropriate buffering to support transfers on the default pipe which are not directly visible to the client such as setting a device address. For those transfers which are directly visible to the client, such as sending vendor and class commands or reading a device descriptor, the client must provide the required buffering.

10.5.1.2.2 Client Pipes

Any pipe not owned and managed by the USBD can be owned and managed by a USBD client. From the USBD viewpoint, the pipe is owned by a single client. In fact, a cooperative group of clients can manage the pipe provided they behave as a single coordinated entity when using the pipe.

The client is responsible for providing the amount of buffering it needs to service the data transfer rate of the pipe within a service interval attainable by the client. Additional buffering requirements for working space are specified by the USB System.

10.5.1.3 Service Capabilities

USBD provides services in the following categories:

- Configuration via Command mechanisms
- Transfer Services via both Command and Pipe mechanisms
- Event Notification
- Status Reporting and Error Recovery

10.5.2 USBD Command Mechanism Requirements

USBD Command mechanisms allow a client generic access to a USB device. Generally these commands allow the client to make read or write accesses to one of potentially several device data and control spaces. The client provides as little as a device identifier and the relevant data or empty buffer pointer.

USBD Command transfers do not require that the USB device be configured. Many of the device configuration facilities provided by USBD are Command transfers.

Following are the specific requirements on the Command mechanisms provided.

10.5.2.1 Interface State Control

USBD clients must be able to set a specified interface to any settable pipe state. Setting an interface state results in all of the pipes in that interface moving to that state. Additionally, all of the pipes in an interface may be reset or aborted.

10.5.2.2 Pipe State Control

USBD pipe state has two components:

- Host status
- Reflected endpoint status

Whenever the pipe status is reported, the value for both components will be identified. The pipe status reflected from the endpoint is the result of the endpoint being in a particular state. The USBD client manages the pipe state as reported by USBD. For any pipe state reflected from the endpoint, the client must also interact with the endpoint to change the state.

A USBD pipe is in exactly one of the following states:

- Active. The pipe's policy has been set and the pipe is able to transmit data. The client can query as to whether any IRPs are outstanding for a particular pipe. Pipes for which there are no outstanding IRPs are still considered to be in the active state as long as they are able to accept new IRPs.
- Stalled. An error has occurred on the pipe. This state may also be a reflection of the corresponding *stalled* endpoint on the device.
- Idle. The pipe will not accept further IRPs. The endpoint must also be in the *idle* state.

USBD clients must be able to set a specified pipe to the idle or active state from any of the states described above. Clients also must be able to set an endpoint to either the active or idle state. Additionally, clients manipulate pipe state in the following ways:

- Aborting a pipe. All of the IRPs scheduled for a pipe are retired immediately and returned to the client with a status indicating they have been aborted. Neither the host state nor the reflected endpoint state of the pipe is affected.
- Resetting a pipe. The pipe's IRPs are aborted. The host state is moved to active. If the reflected endpoint state needs to be changed, that must be commanded explicitly by the USBD client.

10.5.2.3 Getting Descriptors

USBDI must provide a mechanism to retrieve standard device, configuration and string descriptors, as well as any class- or vendor-specific descriptors.

10.5.2.4 Getting Current Configuration Settings

USBDI must provide a facility to return, for any specified device, the current configuration descriptor. If the device is not configured, no configuration descriptor is returned. This action is equivalent to returning the configuration descriptor for the current configuration by requesting the specific configuration descriptor. It does not, however, require the client to know the identifier for the current configuration. This will return all of the configuration information, including:

- All of the configuration descriptor information as stored on the device including all of the alternate settings for all of the interfaces

- Indicators for which of the alternate settings for interfaces are active
- Pipe handles for endpoints in the active alternate settings for interfaces
- Actual *MaxPacketSizes* for endpoints in the active alternate settings for interfaces

Additionally, for any specified pipe, USBDI must provide a facility to return the *MaxPacketSize* which is currently being used by the pipe.

10.5.2.5 Adding Devices

USBDI must provide a mechanism for the hub driver to inform USBD of the addition of a new device to a specified USB and to retrieve the USBD ID of the new USB device. USBD tasks include assigning the device address and preparing the device's default pipe for use.

10.5.2.6 Removing Devices

USBDI must provide a facility for the hub driver to inform USBD that a specific device has been removed.

10.5.2.7 Managing Status

USBDI must provide a mechanism for obtaining and clearing device-based status, on a device, interface, or pipe basis.

10.5.2.8 Sending Class Commands

This USBDI mechanism is used by a client, typically a class specific or adaptive driver, to send one or more class specific commands to a device.

10.5.2.9 Sending Vendor Commands

This USBDI mechanism is used by a client to send one or more vendor specific commands to a device.

10.5.2.10 Establishing Alternate Settings

USBDI must provide a mechanism to change the alternate setting for a specified interface. As a result, the pipe handles for the previous setting are released and new pipe handles for the interface are returned. For this request to succeed, the interface must be idle; i.e., all pipes in the interface must be in the idle state.

10.5.2.11 Establishing a Configuration

Configuring software requests a configuration by passing a buffer containing a configuration descriptor to USBD. USBD requests resources for the endpoints in the configuration, and if all resource requests succeed, USBD sets the device configuration and returns interface handles with corresponding pipe handles for all of the active endpoints. The default values are used for all alternate settings for interfaces and for the *MaxPacketSize* for endpoints. These default values may be subsequently modified.

Note: the specific interface implementing configuration may require specific alternate settings to be identified.

10.5.2.12 Setting Descriptors

For devices supporting this behavior, USBDI allows existing descriptors to be updated or new descriptors to be added.

10.5.2.13 Establishing the Maximum Packet Size for a Pipe

USBDI must provide a mechanism to modify a pipe's transfer characteristics. USBD adjusts requested resources and sets the current maximum data payload size per bus transaction for the specified pipe. This service may only apply to a pipe which has been created by establishing a configuration and which is currently idle.

10.5.3 USBD Pipe Mechanisms

This part of USBDI offers clients the highest-speed, lowest overhead data transfer services possible. Higher performance is achieved by shifting some pipe management responsibilities from USBD to the client. As a result, the Pipe mechanisms are implemented at a more primitive level than the data transfer services provided by the USBD Command mechanisms. Pipe mechanisms do not allow access to a device's default pipe.

USBD Pipe transfers are available only after both the device and USB configuration have completed successfully. At the time the device is configured, USBD requests the resources required to support all device pipes in the configuration. Clients are allowed to modify the configuration, constrained by whether the specified interface or pipe is idle.

Clients provide full buffers to outgoing pipes and retrieve transfer status information following the completion of a request. The transfer status returned for an outgoing pipe allows the client to determine the success or failure of the transfer.

Clients provide empty buffers to incoming pipes and retrieve the filled buffers and transfer status information from incoming pipes following the completion of a request. The transfer status returned for an incoming pipe allows a client to determine the amount and the quality of the data received.

10.5.3.1 Supported Pipe Types

The four types of pipes supported, based on the four transfer types, are described below.

10.5.3.1.1 Isochronous Data Transfers

Each buffer queued for an isochronous pipe is required to be viewable as a stream of samples. As with all Pipe transfers, the client establishes a policy for using this isochronous pipe, including the relevant service interval for this client. Lost or missing bytes, which are detected on input, and transmission problems, which are noted on output, are indicated to the client.

The client queues a first buffer, starting the pipe streaming service. To maintain the continuous streaming transfer model used in all isochronous transfers, the client queues an additional buffer before the current buffer is retired.

USBD is required to be able to provide a sample stream view of the client's data stream. In other words, using the client's specified method of synchronization, the precise packetization of the data is hidden from the client. Additionally, a given transaction is always contained completely within some client data buffer.

For an output pipe, the client provides a buffer of data. USBD allocates the data across the frames for the service period using the client's chosen method of synchronization.

For an input pipe, the client must provide an empty buffer large enough to hold the maximum number of bytes the client's device will deliver in the service period. The USB System strips USB defined packaging information from the stream such that bytes are contiguous in the client's buffer. Where missing or invalid bytes are indicated, USBD leaves the space which the bytes would have occupied in place in the buffer and identifies the error. One of the consequences of using no synchronization method, is that this reserved space is assumed to be the maximum packet size. The buffer-retired notification occurs when the IRP completes. Note that the input buffer need not be full when returned to the client.

USBD may optionally provide additional views of isochronous data streams. USBD is also required to be able to provide a packet stream view of the client's data stream.

10.5.3.1.2 Interrupt Transfers

Interrupt transfers originate in a USB device and are delivered to a client of the USB Driver.

The client queues a buffer large enough to hold the interrupt transfer data (typically a single USB transaction). When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

10.5.3.1.3 Bulk Transfers

Bulk transfers may originate either from the device or the client. No periodicity or guaranteed latency is assumed. When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

10.5.3.1.4 Control Transfers

All message pipes transfer data in both directions. In all cases, the client outputs a setup stage to the device endpoint. The optional data stage may be either an input or an output and the final status is always logically presented to the host. For details of the defined message protocol, refer to Chapter 8.

The client prepares a buffer specifying the command phase and any optional data or empty buffer space. The client receives a buffer-retired notification when all phases of the control transfer are complete, or an error notification, if the transfer is aborted due to transmission error.

10.5.3.2 USBD Pipe Mechanism Requirements

The following Pipe mechanisms are provided.

10.5.3.2.1 Aborting IRPs

USBDI must allow IRPs for a particular pipes to be aborted.

10.5.3.2.2 Adjusting the Start-Of-Frame

USBDI must allow a master client to change the number of bit times in a USB frame. A client wishing to adjust the SOF must already have received master client status from USBD (refer to Section 10.5.3.2.5). Invoking this change more frequently than once every 6 ms has undefined results.

10.5.3.2.3 Managing Pipe Policy

USBDI must allow a client to set and clear the policy for an individual pipe or for an entire interface. Any IRPs made by the client prior to successfully setting a policy are rejected by USBD.

10.5.3.2.4 Queuing IRPs

USBDI must allow clients to queue IRPs for a given pipe. When IRPs are returned to the client, the request status is also returned. A mechanism is provided by USBD to identify a group of isochronous IRPs whose first transactions will all occur in the same frame.

10.5.3.2.5 Being a Master Client

USBDI must allow a client to request becoming a master client for a given USB and to release this capability when it is no longer required. Only master clients may adjust the SOF for a given USB.

A client requesting master status identifies itself with an interface handle for the device from which it is mastering.

10.5.4 Managing the USB via the USBD Mechanisms

Using the provided USBD mechanisms, the following general capabilities are supported by any USB System.

10.5.4.1 Configuration Services

Configuration Services operate on a per device basis. USBD performs device configuration at the direction of the configuring software. A hub driver has a special role in device management and provides at least the following capabilities:

- Device attach/detach recognition, driven by an interrupt pipe owned by the hub driver
- Device reset, accomplished by the hub driver by resetting the hub port upstream of the device
- Directs USBD to perform device address assignment

USBDI additionally provides the following configuration facilities, which may be used by the hub driver or other configuring software available on the host:

- Device identification and access to configuration information (via access to descriptors on the device)
- Device configuration via Command mechanisms

When the hub driver informs USBD of a device attachment, USBD establishes the default pipe for the new device.

10.5.4.1.1 Configuration Management

Configuration Management services are provided primarily as a set of specific interface commands which generate USB transactions on the default pipe. The notable exception is the use of an additional interrupt pipe that delivers hub status directly to the hub driver.

Every hub initiates an interrupt transfer when there is a change in the state of one of the hub ports. Generally, the port state change will be the connection or removal of a downstream USB device. (Refer to Chapter 11 for more information.)

10.5.4.1.2 Initial Device Configuration

The device configuration process begins when a hub reports via its status change pipe the connection of a new USB device.

Configuration Management services allow configuring software to select a USB device configuration from the set of configurations listed in the device. USBD verifies that the data transfer rates given for all endpoints in the configuration do not exceed the capabilities of the USB with the current schedule before setting the device configuration.

10.5.4.1.3 Modifying a Device Configuration

Configuration Management services allow configuring software to replace a USB device configuration with another configuration from the set of configurations listed in the device. The operation succeeds if the data transfer rates given for all end points in the new configuration fit within the capabilities of the USB with the current schedule. If the new configuration is rejected, the previous configuration remains.

Configuration Management services allow configuring software to return a USB device to a not configured state.

10.5.4.1.4 Device Removal

Error recovery and/or device removal processing begins when a hub reports via its status change pipe that communication with a USB device has ceased.

10.5.4.2 Bus and Device Management

Bus and Device Management services allow a client to become the master client on a USB, and as the master client, to adjust the number of bit times in a frame on that bus. A master client can explicitly release master status, or the client's master status will be automatically released when the device containing the referenced interface is reset or detached. There can be at most one master client on a USB. The new master client will be awarded a special master handle to be used when adjusting the SOF.

A master client may add or subtract one bit time to the current USB frame. The client must reference an interface identifying the USB being adjusted and allowing the client's master status to be validated. Adjusting SOF more frequently than once every 6 ms has undefined results.

10.5.4.3 Power Control

The USB System will provide power management in a system-specific manner. No USB requirements beyond those described in Chapter 9 are placed on the USB System.

No standard commands are provided for use with all USB devices. Individual devices may choose to offer a Power Control interface via vendor specific control(s). Device classes may define class-specific Power Control capabilities.

All USB devices must however support the Powered Down state (refer to Chapter 9). Basic control of the power provided to a device is exercised via control of the hub port to which the device is attached.

10.5.4.4 Event Notifications

USBD clients receive several kinds of event notifications through a number of sources:

- Completion of an action initiated by a client.
- Interrupt transfers over stream pipes can deliver notice of device events directly to USBD clients. For example, hubs use an interrupt pipe to deliver events corresponding to changes in hub status.
- Event data can be embedded by devices in streams.
- Standard device interface commands, device class commands, vendor specific commands, and even general control transfers over message pipes can all be used to poll devices for event conditions.

10.5.4.5 Status Reporting and Error Recovery Services

The Command and Pipe mechanisms both provide status reporting on individual requests as they are invoked and completed.

Additionally, USB device status is available to USBD clients using the Command mechanisms.

USBD provides clients with pipe error recovery mechanisms by allowing pipes to be reset or aborted.

10.6 Operating System Environment Guides

As noted previously, the actual interfaces between USB software and host software are specific to the host platform and operating system. A companion specification is required for each combination of platform and operating system with USB support. These specifications describe the specific interfaces used to integrate USB into the host. Each operating system provider for USB software identifies a compatible Universal USB Specification revision.