

# 5

## **Control Transfers: Structured Requests for Critical Data**

Of USB's four transfer types, control transfers have the most complex structure. They're also the only transfer type with functions defined by the USB specification. This chapter looks in greater detail at the structure of control transfers and the requests defined in the specification.

### **Elements of a Control Transfer**

As Chapter 2 explained, control transfers enable the host and a device to exchange information about the device's capabilities. Control transfers also offer a way for devices to transfer other class-specific or vendor-specific information. As explained in Chapter 3, a control transfer has a defined format consisting of a Setup stage, a Data stage (optional for some transfers),

and a Status stage. Each stage consists of one or more transactions that each contain a token phase, data phase, and handshake phase. Each phase transfers a token, data, or handshake packet.

As described in Chapter 2, low-speed transfers also use PRE packets, high-speed transfers use the PING protocol, and some low- and full-speed transfers use split transactions. Each packet also contains error-checking bits. Application programmers, device-driver writers, and firmware developers don't have to worry about PREs, PINGs, error-checking, or split transactions because the host controller, hubs, and device hardware handle these protocols.

### Setup Stage

The Setup stage consists of a Setup transaction, which has two purposes: to identify the transfer as a control transfer and to transmit the request and other information that the device will need to complete the request.

Devices must accept and acknowledge every Setup transaction. A device that is in the middle of another control transfer must abandon that transfer and acknowledge the new Setup transaction. Here are more details about each of the packets in the Setup stage's transaction:

#### Token Packet

**Purpose:** identifies the receiver and identifies the transaction as a Setup transaction.

**Sent by:** the host.

**PID:** SETUP

**Additional Contents:** the device and endpoint addresses.

#### Data Packet

**Purpose:** transmits the request and related information.

**Sent by:** the host.

**PID:** DATA0

**Additional Contents:** eight bytes in five fields: `bmRequestType`, `bRequest`, `wValue`, `wIndex`, and `wLength`.

**bmRequestType** is a byte that specifies the direction of data flow, the type of request, and the recipient.

Bit 7 is a Direction bit that names the direction of data flow for data in the Data stage. Host to device (OUT) or no Data stage is 0; device to host (IN) is 1.

Bits 6 and 5 are Request Type bits that specify whether the request is one of USB's standard requests (00), a request defined for a specific USB class (01), or a request defined by a vendor-specific driver for use with a particular product or products (10).

Bits 4 through 0 are Recipient bits that define whether the request is directed to the device (00000) or to a specific interface (00001), endpoint (00010), or other element (00011) in the device.

**bRequest** is a byte that specifies the request. Every defined request has a unique `bRequest` value. When the Request Type bits in `bmRequestType` = 00, `bRequest` specifies one of the standard USB requests. When the Request Type bits = 01, `bRequest` specifies a request defined for the device's class. When the Request Type bits = 10, `bRequest` specifies a request defined by a vendor.

**wValue** is two bytes that the host may use to pass information to the device. Each request may define the meaning of these bytes in its own way. For example, in a `Set_Address` request, `wValue` contains the device address.

**wIndex** is two bytes that the host may use to pass information to the device. A typical use is to pass an index or offset such as an interface or endpoint number, but each request may define the meaning of these bytes in any way. When passing an endpoint index, bits 0-3 indicate the endpoint number, and bit 7 = 0 for a Control or OUT endpoint or 1 for an IN endpoint. When passing an interface index, bits 0-7 are the interface number. All unused bits are zero.

**wLength** is two bytes containing the number of data bytes in the Data stage that follows. For a host-to-device transfer, `wLength` is the exact number of

bytes the host wants to transfer. For a device-to-host transfer, `wLength` is a maximum, and the device may return this number of bytes or fewer. If the `wLength` field is zero, there is no Data stage.

### **Handshake Packet**

**Purpose:** transmits the device's acknowledgement.

**Sent by:** the device.

**PID:** ACK.

**Additional Contents:** none. The handshake packet consists of the PID alone.

**Comments:** If the device detects an error in the received Setup or Data packet, the device returns no handshake. The device's hardware typically handles the error checking and sending of the ACK, with no programming required.

### **Data Stage**

When a control transfer contains a Data stage, the stage consists of one or more IN or OUT transactions. The device descriptor specifies the maximum number of data bytes in a transaction at Endpoint 0.

When the Data stage uses IN transactions, the device sends data to the host. An example is the `Get_Descriptor` request, where the device sends a requested descriptor to the host. When the Data stage uses OUT transactions, the host sends data to the device. An example is HID-class request `Set_Report`, where the host sends a report to a device. If the `wLength` field in the Setup transaction is zero, there is no Data stage. For example, in the `Set_Configuration` request, the host passes a configuration value to the peripheral in the `wValue` field of the Setup stage's data packet, so there's no need for a Data stage.

If all of the data can't fit in one packet, the stage uses multiple transactions. The number of transactions required to send all of the data for a transfer equals the value in the Setup transaction's `wLength` field divided by the `wMaxPacketSize` value in the endpoint's descriptor, rounded up. For exam-

## Control Transfers: Structured Requests for Critical Data

ple, in a Get\_Descriptor request, if wLength is 18 and wMaxPacketSize is 8, the transfer requires 3 Data transactions. The transactions in the Data stage must all be in the same direction. When the Data stage is present but there is no data to transfer, the data phase consists of a zero-length data packet (the PID only).

The host uses split transactions in the Data stage when the device is low or full speed and an upstream hub connects to a high-speed bus. The host may use the PING protocol when the device is high speed, the Data stage uses OUT transactions, and there is more than one data transaction.

Each IN or OUT transaction in the Data stage contains token, data, and handshake packets. Here are more details about each of the packets in the Data stage's transaction(s):

### Token Packet

**Purpose:** identifies the receiver and identifies the transaction as an IN or OUT transaction.

**Sent by:** the host.

**PID:** if the request requires the device to send data to the host, the PID is IN. If the request requires the host to send data to the device, the PID is OUT.

**Additional Contents:** the device and endpoint addresses.

### Data Packet

**Purpose:** transfers all or a portion of the data specified in the wLength field of the Setup transaction's data packet.

**Sent by:** if the token packet's PID is IN, the device sends the data packet; if the token packet's PID is OUT, the host sends the data packet.

**PID:** The first packet is DATA1. Any additional packets in the Data stage alternate DATA0/DATA1.

**Additional Contents:** the data or a zero-length data packet.

### Handshake Packet

**Purpose:** the data packet's receiver returns status information.

**Sent by:** the receiver of the Data stage's data packet. If the token packet's PID is IN, the host sends the handshake packet. If the token packet's PID is OUT, the device sends the handshake packet.

**PID:** Any device may return ACK (valid data was received), NAK (the endpoint is busy), or STALL (the request isn't supported or the endpoint is halted). A high-speed device that is receiving multiple data packets may return NYET (the current transaction's data was accepted but the endpoint isn't yet ready for another data packet). The host can return only ACK.

**Additional Contents:** None. The handshake packet consists of the PID alone.

**Comments:** If the receiver detected an error in the token or data packet, the receiver returns no handshake packet.

### Status Stage

The Status stage is where the device reports the success or failure of the entire transfer. The purpose of the Status stage is similar to the purpose of a transaction's handshake packet, and in fact the status information sometimes travels in the handshake packet of the Status stage. But the Status stage reports the success or failure of the entire transfer, rather than of a single transaction.

In some cases (such as after receiving the first packet of a device descriptor during enumeration), the host may begin the Status stage before the Data stage has completed, and the device must detect the token packet of the Status stage, abandon the Data stage, and complete the Status stage.

Here are more details about each of the packets in the Status stage's transaction:

#### Token Packet

**Purpose:** identifies the receiver and indicates the direction of the Status stage's data packet.

## Control Transfers: Structured Requests for Critical Data

**Sent by:** the host.

**PID:** the opposite of the direction of the previous transaction's data packet. If the Data stage's PID was OUT or if there was no Data stage, the Status stage's PID is IN. If the Data stage's PID was IN, the Status stage's PID is OUT.

**Additional Contents:** the device and endpoint addresses.

### Data Packet

**Purpose:** enables the receiver of the Data stage's data to indicate the status of the transfer.

**Sent by:** if the Status stage's token packet's PID is IN, the device sends the data packet; if the Status stage's token packet's PID is OUT, the host sends the data packet.

**PID type:** DATA1

**Additional Contents:** The host sends a zero-length data packet. A device may send a zero-length data packet (success), NAK (busy), or STALL (endpoint halted).

**Comments:** For most requests, a zero-length data packet sent by the device indicates that the requested action (if any) has been taken. An exception is Set\_Address, which the device implements after the Status stage has completed.

### Handshake Packet

**Purpose:** the sender of the Data stage's data indicates the status of the transfer.

**Sent by:** the receiver of the Status stage's data packet. If the Status stage's token packet's PID is IN, the host sends the handshake packet; if the token packet's PID is OUT, the device sends the data packet.

**PID type:** the device's response may be ACK (success), NAK (busy), or STALL (the request isn't supported or the endpoint is halted). The host's response to a data packet received without error must be ACK.

**Additional Contents:** none. The handshake packet consists of the PID alone.

**Comments:** The Status stage's handshake packet is the final transmission in the transfer. If the receiver detected an error in the token or data packet, the receiver returns no handshake packet.

For any request that's expected to take many milliseconds to carry out, the protocol should define an alternate way to determine when the request has completed. Doing so ensures that the host doesn't waste a lot of time asking for an acknowledgement that will take a long time to appear. An example is the `Set_Port_Feature(PORT_RESET)` request sent to a hub. The reset signal lasts at least 10 milliseconds. Rather than forcing the host to wait this long for the device to complete the reset, the hub acknowledges receiving the request when the hub first places the port in the reset state. When the reset is complete, the hub sets a bit that the host can retrieve at its leisure, using a `Get_Port_Status` request.

## Handling Errors

Devices don't always carry out every control-transfer request they receive. The device's firmware might not support a request. Or the device may be unable to respond because its firmware has crashed, or the endpoint is in the Halt condition, or the device is no longer attached to the bus. The host may also decide for any reason to end a transfer early, before all of the data has been sent.

An example of an unsupported request is one that uses a request code that the device's firmware doesn't know how to respond to. Or a device may support the request but other information in the Setup stage doesn't match what the device expects or supports. On these occasions, a Request Error condition exists and the device notifies the host by sending a STALL code in a handshake packet. Devices must respond to the Setup transaction with an ACK, so the STALL must transmit in a handshake packet in the Data or Status stage.

On failing to get an expected response or on detecting an error in received data or a Halt condition at the endpoint, the host abandons the transfer.

## Control Transfers: Structured Requests for Critical Data

The host then tries to re-establish communications by sending the token packet for a new Setup transaction. If a device receives a token packet for a Setup transaction before completing a previous control transfer, the device must abandon the previous transfer and begin the new one. If the transfer is using the Default Control Pipe and a new token packet doesn't cause the device to recover, the host takes more drastic action, requesting the device's hub to reset the device's port.

The host may also end a transfer early by beginning the Status stage before completing all of the Data stage's transactions. In this case, the device must abandon the rest of the data and respond to the Status stage the same as if all of the data had transferred.

### Device Firmware

The following descriptions are an overview of what typical device firmware must do to support control transfers.

#### Control Write Requests with a Data Stage

To complete a Control Write request where the host sends data to the device, the device must detect the request in the Setup stage, receive data in the Data stage, and return a handshake in the Status stage:

1. The hardware detects a Setup packet, stores the contents of the transaction's data packet, returns ACK, and triggers an interrupt.
2. The interrupt-service routine decodes the request and configures Endpoint 0 to accept data that arrives following an OUT token packet. The endpoint should also be able to handle the arrival of a new Setup packet, in case the host decides to abandon the transfer early.
3. The device returns to normal operation. The arrival of an OUT token packet at Endpoint 0 indicates that the host is sending data in the Data stage. The endpoint returns ACK in the handshake packet and the hardware triggers an interrupt.
4. The interrupt-service routine stores or uses the received data.

5. If more data packets are expected in the Data stage, steps 3 and 4 repeat for additional OUT transactions, up to the wLength value in the Setup transaction.
6. When all of the data has been received, the firmware configures Endpoint 0 to send a zero-length data packet in response to an IN token packet. The host returns ACK to complete the transfer.

### **Control Write Requests with No Data Stage**

To complete a Control Write request when there is no Data stage, the device must detect the request in the Setup stage and send a handshake in the Status stage:

1. The hardware detects a Setup packet, stores the contents of the transaction's data packet, returns ACK, and triggers an interrupt.
2. The interrupt-service routine decodes the request, does what is needed to perform the requested action, and configures Endpoint 0 to respond to an IN token packet. The endpoint should also be able to handle the arrival of a new Setup packet in case the host decides to abandon the transfer early.
3. A received IN token packet begins the Status stage. The endpoint sends a zero-length data packet and the host returns ACK to complete the transfer.

### **Control Read Requests**

To complete a Control Read request, where the host requests data from the device, the device must detect the request in the Setup stage, send data in the Data stage, and acknowledge a received handshake in the Status stage:

1. The hardware detects a Setup packet, stores the contents of the transaction's data packet, returns ACK, and triggers an interrupt.
2. The interrupt-service routine decodes the request and configures Endpoint 0 to send data on receiving an IN token packet. The endpoint should also be able to handle the arrival of a new Setup or OUT packet in case the host decides to abandon the transfer or begin the Status stage early.
3. The device returns to normal operation. The arrival of an IN token packet at Endpoint 0 indicates that the host is requesting data in the Data

stage. The device hardware sends the data, detects the received ACK from the host, and triggers an interrupt.

4. If there is more data to send, the interrupt service routine configures the endpoint to send the data on receiving another IN token packet and steps 3 and 4 repeat.

5. On receiving an OUT token packet followed by a zero-length data packet, the endpoint returns ACK to complete the transfer.

## The Requests

Table 5-1 summarizes USB's eleven standard requests. Following the table is more information about each request. All devices must respond to these requests (though the response may be just a STALL). The values range from 00 to 0Ch, with some values unused.

Most of the requests are in pairs, with each Set request having a corresponding Get or Clear request. The exceptions are Set\_Address, Synch\_Frame, and Get\_Status.

Table 5-1: The USB specification defines eleven standard requests for Control transfers.

Request Number	Request	Data source (Data stage)	Recipient	wValue	wIndex	Data Length (bytes) in Data stage (wLength)	Data (in Data stage)
00h	Get_Status	device	device, interface, endpoint	0	device, interface, or endpoint	2	status
01h	Clear_Feature	no Data stage	device, interface, endpoint	feature	device, interface, or endpoint	–	–
03h	Set_Feature	no Data stage	device, interface, endpoint	feature	device, interface, or endpoint	–	–
05h	Set_Address	no Data stage	device	device address	0	–	–
06h	Get_Descriptor	device	device	descriptor type and index	device or language ID	descriptor length	descriptor
07h	Set_Descriptor	host	device	descriptor type and index	device or language ID	descriptor length	descriptor
08h	Get_Configuration	device	device	0	device	1	configuration
09h	Set_Configuration	no Data stage	device	configuration	device	–	–
0Ah	Get_Interface	device	interface	0	interface	1	alternate setting
0Bh	Set_Interface	no Data stage	interface	interface	interface	–	–
0Ch	Synch_Frame	device	endpoint	0	endpoint	2	frame number

### Get\_Status

**Purpose:** The host requests the status of the features of a device, interface, or endpoint.

**Request Number (bRequest):** 00h

**Source of Data:** device

**Data Length (wLength):** 2

**Contents of wValue field:** 0

**Contents of wIndex field:** For a device, 0. For an interface, the interface number. For an endpoint, the endpoint number.

**Contents of data packet in the Data stage:** the device, interface, or endpoint status.

**Supported states:** Default: undefined. Address: OK for address 0, Endpoint 0. Otherwise the device returns a STALL. Configured: OK.

**Behavior on error:** The device returns a STALL if the interface or endpoint doesn't exist.

**Comments:** For requests directed to the device, two status bits are defined. Bit 0 is the Self-Powered field: 0=bus-powered, 1=self-powered. (The host can't change this value.) Bit 1 is the Remote Wakeup field. The default on reset is 0 (disabled). All other bits are reserved. For requests directed to an interface, all bits are reserved. For requests directed to an endpoint, only bit 0 is defined. Bit 0=1 indicates a Halt condition. See Set\_Feature and Clear\_Feature for more details on Remote Wakeup and Halt.

## Clear\_Feature

**Purpose:** The host requests to disable a feature on a device, interface, or endpoint.

**Request Number (bRequest):** 01h.

**Source of Data:** no Data stage

**Data Length (wLength):** none

**Contents of wValue field:** the feature to disable

**Contents of wIndex field:** For a device feature, 0. For an interface feature, the interface number. For an endpoint feature, the endpoint number.

**Contents of data packet in the Data stage:** none.

**Supported states:** Default: undefined. Address: OK for address 0, Endpoint 0. Otherwise the device returns a STALL. Configured: OK.

**Behavior on error:** If the feature, device, or endpoint specified doesn't exist, or if the feature can't be cleared, the device responds with a STALL. Behavior is undefined when wLength is greater than 0.

**Comments:** This request can clear the DEVICE\_REMOTE\_WAKEUP feature and the ENDPOINT\_HALT feature, but not the TEST\_MODE feature. Clear\_Feature(ENDPOINT\_HALT) resets the endpoint's data toggle to DATA0. See also Set\_Feature and Get\_Status.

### Set\_Feature

**Purpose:** The host requests to enable a feature on a device, interface, or endpoint.

**Request Number (bRequest):** 03h

**Source of Data:** no Data stage

**Data Length (wLength):** none

**Contents of wValue field:** the feature to enable

**Contents of wIndex field:** For a device, 0. For an interface, the interface number. For an endpoint, the endpoint number.

**Contents of data packet in the Data stage:** none.

**Supported states:** For features other than TEST\_MODE: Default: undefined. Address: OK for address 0, Endpoint 0. Otherwise the device returns a STALL. Configured: OK. The TEST\_MODE feature must be supported when using high speed in the Default, Address, and Configured states.

**Behavior on error:** If the endpoint or interface specified doesn't exist, the device responds with a STALL.

**Comments:** The USB 2.0 specification defines three features. ENDPOINT\_HALT, with a value of 0, applies to endpoints. Bulk and interrupt endpoints must support the Halt condition. Two types of events may cause a Halt condition: a communications problem such as the device's not receiving a handshake packet or receiving more data than expected, or the device's receiving a Set\_Feature request to halt the endpoint. DEVICE\_REMOTE\_WAKEUP, with a value of 1, applies to devices. When the host sets the DEVICE\_REMOTE\_WAKEUP feature, a device in the Suspend state can request the host to resume communications. TEST\_MODE, with a value of 2, applies to devices. Setting this feature causes an the upstream-facing port to enter a test mode. Chapter 18 has more about test mode.

The Get\_Status request tells the host what features, if any, are enabled. Also see Clear\_Feature.

## Set\_Address

**Purpose:** The host specifies an address to use in future communications with the device.

**Request Number (bRequest):** 05h

**Source of Data:** no Data stage

**Data Length (wLength):** none

**Contents of wValue field:** new device address. Allowed values are 1 through 127. Each device on the bus, including the root hub, has a unique address.

**Contents of wIndex field:** 0

**Contents of data packet in the Data stage:** none

**Supported States:** Default, Address.

**Behavior on error:** not specified.

**Comments:** When a hub enables a port after power-up or attachment, the port uses the default address of 0 until completing a Set\_Address request from the host.

This request is unlike most other requests because the device doesn't carry out the request until the device has completed the Status stage of the request by sending a zero-length data packet. The host sends the Status stage's token packet to the default address, so the device must detect and respond to this packet before changing its address.

After completing this request, all communications use the new address.

A device using the default address of zero is in the Default state. After completing a Set\_Address request to set an address other than zero, the device enters the Address state.

A device must send the handshake packet within 50 milliseconds after receiving the request and must implement the request within 2 milliseconds after completing the Status stage.

### Get\_Descriptor

**Purpose:** The host requests a specific descriptor.

**Request Number (bRequest):** 06h

**Source of Data:** device

**Data Length (wLength):** the number of bytes to return. If the descriptor is longer than wLength, the device returns up to wLength bytes. If the descriptor is shorter than wLength, the device returns the descriptor. If the descriptor is shorter than wLength and an even multiple of the endpoint's maximum packet size, the device follows the descriptor with a zero-length data packet. The host detects the end of the data on receipt of either the requested amount of data or a data packet containing less than the maximum packet size (including zero bytes).

**Contents of wValue field:** High byte: descriptor type. Low byte: descriptor index, to specify which descriptor to return when there are multiple descriptors of the same type.

**Contents of wIndex field:** for String descriptors, Language ID. Otherwise zero.

**Contents of data packet in the Data stage:** the requested descriptor.

**Supported states:** Default, Address, Configured.

**Behavior on error:** When a device receives a request that the device doesn't support, the device should return a STALL.

**Comments:** A host can request the following descriptor types: device, device\_qualifier, configuration, other\_speed configuration, and string. On receiving a request for a configuration or other\_speed configuration descriptor, the device should return the requested descriptor followed by all of the configuration's subordinate descriptors up to the number of bytes requested. A class or vendor can also define descriptors that the host can request, such as the HID-class report descriptor. See also Set\_Descriptor.

## Set\_Descriptor

**Purpose:** The host adds a descriptor or updates an existing descriptor.

**Request Number (bRequest):** 07h

**Source of Data:** host

**Data Length (wLength):** The number of bytes the host will transfer to the device.

**Contents of wValue field:** high byte: descriptor type. (See Get\_Descriptor) Low byte: descriptor index to specify which descriptor is being sent when there are multiple descriptors of the same type.

**Contents of wIndex field:** For string descriptors, Language ID. Otherwise zero.

**Contents of data packet in the Data stage:** descriptor length.

**Supported states:** Address and Configured.

**Behavior on error:** When a device receives a request that the device doesn't support, the device should return a STALL.

**Comments:** This request makes it possible for the host to add new descriptors or change an existing descriptor. Many devices don't support this request because it allows errant software to place incorrect information in a descriptor. See also Get\_Descriptor.

## Get\_Configuration

**Purpose:** The host requests the value of the current device configuration.

**Request Number (bRequest):** 08h

**Source of Data:** device

**Data Length (wLength):** 1

**Contents of wValue field:** 0

**Contents of wIndex field:** 0

**Contents of data packet in the Data stage:** Configuration value

**Supported states:** Address (returns zero), Configured

**Behavior on error:** not specified.

**Comments:** A device that isn't configured returns zero. See also Set\_Configuration.

## Set\_Configuration

**Purpose:** The host requests the device to use the specified configuration.

**Request Number (bRequest):** 09h

**Source of Data:** no Data stage

**Data Length (wLength):** none

**Contents of wValue field:** The lower byte specifies a configuration. If the value matches a configuration supported by the device, the device implements the requested configuration. A value of zero indicates not configured. If the value is zero, the device enters the Address state and requires a new Set\_Configuration request to be configured.

**Contents of wIndex field:** 0

**Contents of data packet in the Data stage:** none

**Supported states:** Address, Configured.

**Behavior on error:** If wValue isn't equal to zero or a configuration supported by the device, the device returns a STALL.

**Comments:** After completing a Set\_Configuration request specifying a supported configuration, the device enters the Configured state. Many standard requests require the device to be in the Configured state. See also Get\_Configuration. This request resets the endpoint's data toggle to DATA0.

### Get\_Interface

**Purpose:** For devices with interfaces that have multiple, mutually exclusive settings for an interface, the host requests the currently active interface setting.

**Request Number (bRequest):** 0Ah

**Source of Data:** device

**Data Length (wLength):** 1

**Contents of wValue field:** 0

**Contents of wIndex field:** interface number

**Contents of data packet in the Data stage:** the current setting

**Supported states:** Configured

**Behavior on error:** If the interface doesn't exist, the device returns a STALL.

**Comments:** The interface number in the wIndex field of this request refers to the bInterface field in an interface descriptor. This value distinguishes an interface from other interfaces that are active at the same time. The setting in the Data field in this request refers to the value in the bAlternateInterface field in the interface descriptor. This value identifies which of two or more alternate (mutually exclusive) settings an interface is currently using. Each setting has an interface descriptor and optional endpoint descriptors. Many devices support only one interface setting. See also Set\_Interface.

## Set\_Interface

**Purpose:** For devices with interfaces that have alternate (mutually exclusive) settings, the host requests the device to use a specific interface setting.

**Request Number (bRequest):** 0Bh

**Source of Data:** no Data stage

**Data Length (wLength):** none

**Contents of wValue field:** alternate setting to select

**Contents of wIndex field:** interface number

**Contents of data packet in the Data stage:** none

**Supported states:** Configured

**Behavior on error:** If the device supports only a default interface, the device may return a STALL. If the requested interface or setting doesn't exist, the device returns a STALL.

**Comments:** This request resets the endpoint's data toggle to DATA0. See also Get\_Interface

### Synch\_Frame

**Purpose:** The device sets and reports an endpoint's synchronization frame.

**Request Number (bRequest):** 0Ch

**Source of Data:** host

**Data Length (wLength):** 2

**Contents of wValue field:** 0

**Contents of wIndex field:** endpoint number

**Contents of data packet in the Data stage:** frame number

**Supported states:** Default: undefined. Address: The device returns STALL. Configured: OK.

**Behavior on error:** If the endpoint doesn't support the request, the endpoint should return STALL.

**Comments:** In isochronous transfers, a device endpoint may request data packets that vary in size, following a sequence. For example, an endpoint may send a repeating sequence of 8, 8, 8, 64 bytes. The Synch\_Frame request enables the host and endpoint to agree on which frame will begin the sequence.

On receiving a Synch\_Frame request, an endpoint returns the number of the frame that will precede the beginning of a new sequence

This request is rarely used because there is rarely a need for the information it provides.

## Other Control Requests

In addition to the requests defined in the USB 2.0 specification, a device may respond to class-specific and vendor-specific control requests.

### Class-specific Requests

A class may define requests for devices in its class. A class-specific request may be required or optional for devices in the class. Some requests are unrelated to the standard requests, while others build on the standard requests by defining class-specific fields in a request. An example of a request that's unrelated to the standard requests is the `Get_Max_LUN` request supported by some mass-storage devices. The host uses this request to find out the number of logical units the interface supports. An example of a request that builds on an existing request is the `Get_Port_Status` request that hubs must support. This request is structured like the standard `Get_Status` request. But `Get_Port_Status` has different values in two fields. In `bmRequestType`, bits 6 and 5 are 01 to indicate that the request is defined by a standard USB class, and bits 4 through 0 are 00011 to indicate that the request applies to a unit other than the device or an interface or endpoint. (The request applies to a port on a hub.) The `wIndex` field holds the port number.

### Vendor-specific Requests

A vendor may define custom requests for control transfers with specific devices. Implementing a custom request in a control transfer requires all of the following:

- Vendor-defined fields as needed in the Setup and optional Data stages of the request. Bits 6 and 5 in the Setup stage's data packet are set to 10 to indicate a vendor-defined request.
- In the device, code that detects the request number in the Setup packet and knows how to respond.
- In the host, a vendor-specific device driver to initiate the request. Applications can't initiate vendor-specific requests on their own. The application must call a function exposed by a driver that defines the request.