



ver1.1 - April 10, 2003

## CEUSB2 DRIVER SPECIFICATIONS

CeUsb2 Driver Specifications Document for  
Software Programmers

## CONTENTS

<b>CONTENTS</b> .....	<b>1</b>
<b>1 INTRODUCTION</b> .....	<b>2</b>
1.1 About This Documentation.....	2
1.2 Prerequisites .....	2
1.3 Distribution .....	3
1.4 Overview .....	3
<b>2 DRIVER PROGRAMMING INTERFACE</b> .....	<b>4</b>
2.1 Opening and closing the driver .....	4
2.2 I/O Control (IOCTL) Codes Interface .....	6
<b>3 PROGRAMMING INTERFACE REFERENCE</b> .....	<b>8</b>
3.1 I/O Control (IOCTL) Codes Reference.....	8
3.1.1 IOCTL_CEUSB2_GET_DRIVER_INFORMATION .....	8
3.1.2 IOCTL_CEUSB2_DESCRIPTOR_REQUEST .....	8
3.1.3 IOCTL_CEUSB2_GET_INTERFACE_DESCRIPTOR.....	9
3.1.4 IOCTL_CEUSB2_RESET .....	9
3.1.5 IOCTL_CEUSB2_RESET_PIPE .....	10
3.1.6 IOCTL_CEUSB2_ABORT_PIPE .....	10
3.1.7 IOCTL_CEUSB2_FEATURE_REQUEST .....	10
3.1.8 IOCTL_CEUSB2_VENDOR_OR_CLASS_REQUEST .....	10
3.1.9 IOCTL_CEUSB2_VENDOR_REQUEST .....	11
3.1.10 IOCTL_CEUSB2_GET_LAST_USBD_ERROR.....	11
3.1.11 IOCTL_CEUSB2_GET_CURRENT_FRAME_NUMBER .....	11
3.1.12 IOCTL_CEUSB2_GET_STATUS .....	12
3.1.13 IOCTL_CEUSB2_SET_DEVICE_POWER_STATE.....	12
3.1.14 IOCTL_CEUSB2_GET_DEVICE_POWER_STATE .....	12
3.1.15 IOCTL_CEUSB2_GET_CONFIGURATION.....	12
3.1.16 IOCTL_CEUSB2_SELECT_CONFIGURATION.....	13
3.1.17 IOCTL_CEUSB2_GET_INTERFACE_INFORMATION .....	13
3.1.18 IOCTL_CEUSB2_GET_INTERFACE .....	13
3.1.19 IOCTL_CEUSB2_SELECT_INTERFACE.....	14
3.1.20 IOCTL_CEUSB2_READ_ISO.....	14
3.1.21 IOCTL_CEUSB2_WRITE_ISO .....	14
3.1.22 IOCTL_CEUSB2_START_ISO_STREAM .....	15
3.1.23 IOCTL_CEUSB2_STOP_ISO_STREAM .....	15
3.1.24 IOCTL_CEUSB2_READ_ISO_STREAM.....	15
3.1.25 IOCTL_CEUSB2_READ_BULK.....	15

3.1.26	IOCTL_CEUSB2_WRITE_BULK.....	16
3.1.27	IOCTL_CEUSB2_READ_CONTROL.....	16
3.1.28	IOCTL_CEUSB2_WRITE_CONTROL.....	16
3.2	Interface Structures.....	16
3.2.1	USB_DEVICE_DESCRIPTOR.....	16
3.2.2	USB_CONFIGURATION_DESCRIPTOR.....	17
3.2.3	USB_INTERFACE_DESCRIPTOR.....	18
3.2.4	USB_ENDPOINT_DESCRIPTOR.....	19
3.2.5	USB_STRING_DESCRIPTOR.....	20
3.2.6	USB_COMMON_DESCRIPTOR.....	20
3.2.7	USB_INTERFACE_INFORMATION.....	20
3.2.8	USB_PIPE_INFORMATION.....	21
3.2.9	CEUSB2_DRIVER_INFO.....	22
3.2.10	CEUSB2_DESCRIPTOR_INFO.....	22
3.2.11	CEUSB2_INTERFACE_DESCRIPTOR_INFO.....	23
3.2.12	CEUSB2_INTERFACE_INFO.....	23
3.2.13	CEUSB2_VENDOR_OR_CLASS_REQUEST_CONTROL.....	23
3.2.14	CEUSB2_VENDOR_REQUEST_CONTROL.....	24
3.2.15	CEUSB2_GET_STATUS_INFO.....	24
3.2.16	CEUSB2_FEATURE_REQUEST_INFO.....	24
3.2.17	CEUSB2_POWER_INFO.....	25
3.2.18	CEUSB2_ISO_TRANSFER_INFO.....	25
3.2.19	CEUSB2_ISO_TRANSFER_INFO_EX.....	26
3.2.20	CEUSB2_CONTROL_TRANSFER_INFO.....	26
3.3	Interface Enumeration Types.....	26
3.3.1	USB_PIPE_TYPE.....	26
3.3.2	CEUSB2_REQUEST_RECIPIENT.....	27
3.3.3	CEUSB2_POWER_STATE.....	27

## 1 INTRODUCTION

### 1.1 About This Documentation

CeUsb2 devices have two drivers which are loaded by the system sequentially. This document is intended to give the specifications for the second one, the main driver. This worker driver has the ability to handle general USB functionality, USB data transfers and communicate with CeUsb2 devices through the firmware. Before reading this document it would be helpful to check general CeUsb2 design guide documentation, CeUsb2dg.pdf, to understand the components which comprise the CeUsb2 software which runs in corporation with the driver.

### 1.2 Prerequisites

The CeUsb2 driver interface is accessible directly from C and C++ programming languages. The example codes given in this documentation is written in these languages. Therefore, an intermediate or upper knowledge of C/C++ and Windows programming (Win 32 SDK) is necessary to follow this documentation.

CeUsb2 boards are USB2.0 devices, so it is necessary to understand basics of the USB protocol in order to understand the USB communication and data transfer operations.

### **1.3 Distribution**

After you install CeUsb2 software, you can find the header file for the CeUsb2 driver programming interface header files in CeUsb2\inc directory. These files are:

CeUsb2If.h: Main interface definition header file for CeUsb2 driver (IOCTL codes are in this file).

CeError.h: Contains status and error code definitions of the driver (This file also includes the status codes defined by the loader driver and the CeUsb2 application programming interface).

Guids.h: Guid definitions for CeUsb2 programming interface.

Usbd.h: USB definitions header file.

UsbDef.h : USBD definitions header file.

These files include all IOCTL codes, structures and enumeration types to access the CeUsb2 main driver.

CeUsb driver's executable is CeUsb2.sys which you can find in the driver directory.

### **1.4 Overview**

CeUsb2 main driver (CeUsb2.sys) is developed with WDM model of WIN DDK. It supports almost all PNP requests, handles power management and participates in WMI (Windows management and instrumentation). CeUsb2 driver defines an interface for user mode applications with Input/Output Control (IOCTL) codes, enumerations and structures to control CeUsb2 hardware and perform vendor and class requests, control, bulk and isochronous transfers and many other USB functionalities.

Chapter 2 explains the usage of the driver IOCTL interface with some example codes.

Chapter 3 is a reference for the CeUsb2 driver's programming interface. It explains the IOCTL codes together with the structures and enumeration types used with them.

## 2 DRIVER PROGRAMMING INTERFACE

All User mode access to the CeUsb2 is through I/O Control (IOCTL) calls. A user mode application first gets a handle to the device driver via a call to the Win32 function `CreateFile`. The user mode application then uses the Win32 function `DeviceIoControl` to submit an I/O control code and related input and output buffers to the driver with the handle returned by `CreateFile`.

### 2.1 *Opening and closing the driver*

CeUsb2 devices define device interfaces unlike the old dos device naming technique, to enable user mode applications opening and closing handles through them. This device interface is accessible with a 128 bit GUID identifier. CeUsb2 GUID definitions can be found in `inc\guids.h` file.

Code example 2.1 implements two functions `OpenCeUsb2` and `CloseCeUsb2` to show how CeUsb2 devices are opened and closed. `OpenCeUsb2` searches the system for a CeUsb2 device with a given device index and a device interface GUID. For this process, `SetupDiGetClassDevs`, `SetupDiEnumDeviceInterfaces` and `SetupDiGetDeviceInterfaceDetail` functions from setup API of Win32 SDK are used. Check Win 32 SDK reference documentation to learn more about these functions.

After enumeration if the CeUsb2 device with the given index is found in the system, it is opened with `CreateFile` function. If `CreateFile` function succeeds, it returns a 32 bit device handle which is used as a parameter to access the device with IOCTL codes and close it after all I/O processing is done.

`CloseCeUsb2` function simply uses the `CloseHandle` function of the Win32 SDK to close the previously opened device handle.

```
// Example code 2.1
// Opening and closing CeUsb2 devices
//
#include <Setupapi.h> // Win32 setup API header file, required library file is Setupapi.lib
//
// Opens the CeUsb2 device with the given index and device interface GUID, and stores its handle in
// DevHandle pointer. This functions return 32 bit status code (0 = success).
//
DWORD OpenCeUsb2(GUID Guid, UINT DeviceIndex, HANDLE *DevHandle)
```

```

{
    DWORD dwStatus = 0; // return status
    SP_INTERFACE_DEVICE_DATA DevInterfaceData;
    PSP_INTERFACE_DEVICE_DETAIL_DATA pDevDetail = NULL;
    SP_DEVINFO_DATA DevInfoData;
    DWORD Size;

    // open a device enumeration handle
    HDEVINFO hDevInfo = SetupDiGetClassDevs(
        &Guid, NULL, NULL, DIGCF_PRESENT | DIGCF_INTERFACEDevice );
    if (hDevInfo == INVALID_HANDLE_VALUE)
        return GetLastError();

    // find the device with DeviceIndex index
    DevInterfaceData.cbSize = sizeof(SP_INTERFACE_DEVICE_DATA);
    if(!SetupDiEnumDeviceInterfaces(
        hDevInfo, NULL, &Guid, DeviceIndex, &DevInterfaceData ))
    {
        dwStatus = GetLastError();
        goto exit_open;
    }

    // first get the length of detailed information
    SetupDiGetDeviceInterfaceDetail(
        hDevInfo, &DevInterfaceData, NULL, 0, &Size, NULL );

    pDevDetail = (PSP_INTERFACE_DEVICE_DETAIL_DATA)malloc(Size);
    pDevDetail->cbSize = sizeof(SP_INTERFACE_DEVICE_DETAIL_DATA);
    DevInfoData.cbSize = sizeof(SP_DEVINFO_DATA);

    // get device details
    if(!SetupDiGetDeviceInterfaceDetail(
        hDevInfo, &DevInterfaceData, pDevDetail, Size, NULL, &DevInfoData))
    {
        dwStatus = GetLastError();
        goto exit_open;
    }

    // open the device
    *DevHandle = CreateFile(
        pDevDetail->DevicePath, // name of the device
        GENERIC_READ | GENERIC_WRITE, // access mode
        0, // share mode
        NULL, // security desc.
        OPEN_EXISTING, // how to create
        0, // file attributes
        NULL // template file
    );
    if ( *DevHandle == INVALID_HANDLE_VALUE )
        dwStatus = GetLastError();

exit_open:
    if(pDevDetail)
    {
        free(pDevDetail);
    }
}

```

```

        pDevDetail = NULL;
    }

    SetupDiDestroyDeviceInfoList(hDevInfo); // destroy device information list

    return dwStatus;
}

//
// CloseCeUsb2 function traverses the functionality of OpenCeUsb function, and closes (nullifies) the
// device handle stored during open device process.
//
void CloseCeUsb2(HANDLE DevHandle)
{
    if(!DevHandle)
    {
        CloseHandle(DevHandle);
        DevHandle = NULL;
    }
}

```

## 2.2 I/O Control (IOCTL) Codes Interface

The I/O Control requests are submitted to the driver using the Win32 SDK function DeviceIoControl. The DeviceIoControl function is defined in SDK reference documentation as follows:

```

BOOL DeviceIoControl(
    HANDLE          hDevice,           // handle to device
    DWORD          dwIoControlCode,   // operation
    LPVOID         lpInBuffer,        // input data buffer
    DWORD          nInBufferSize,     // size of input data buffer
    LPVOID         lpOutBuffer,       // output data buffer
    DWORD          nOutBufferSize,    // size of output data buffer
    LPDWORD        lpBytesReturned,   // byte count
    LPOVERLAPPED  lpOverlapped       // overlapped information
);

```

This function returns a nonzero value (TRUE) if it succeeds; otherwise it returns 0 (FALSE). Like most other Windows functions, the reason for the error can be retrieved with GetLastError function.

Following is the explanations of the DeviceIoControl function's parameters. Section 3.1, I/O Control Code (IOCTL) Reference, describes the I/O Control codes that can be passed to this function's dwIoControlCode parameter together with the usage of lpInBuffer, nInBufferSize, lpOutBuffer and nOutBufferSize arguments.

hDevice - Handle to the device on which to perform the operation. This handle is returned by CreateFile function.

dwIoControlCode – I/O control code (IOCTL) for the operation. This value identifies the specific operation to be performed and the type of device on which to perform it. (See Section 3.1).

lpInBuffer – Input data buffer (IOCTL specific).

nInBufferSize – Input data buffer size in bytes (IOCTL specific).

lpOutBuffer - Output data buffer (IOCTL specific).

nOutBufferSize - Output data buffer size in bytes (IOCTL specific).

lpBytesReturned - Pointer to a variable that receives the size, in bytes, of the data transferred by this IOCTL operation. This field is set by the CeUsb2 driver after a successful I/O operation.

lpOverlapped - Pointer to an OVERLAPPED structure. If hDevice was opened with the FILE\_FLAG\_OVERLAPPED flag, lpOverlapped must point to a valid OVERLAPPED structure. In this case, the operation is performed as an overlapped (asynchronous) operation. If the device was opened with FILE\_FLAG\_OVERLAPPED and lpOverlapped is NULL, the function fails in unpredictable ways.

If hDevice was opened without specifying the FILE\_FLAG\_OVERLAPPED flag, lpOverlapped is ignored and DeviceIoControl does not return until the operation has been completed, or an error occurs.

Code example 2.2 shows the simple usage of the CeUsb2 IOCTL interface.

```
// Code example 2.2
// IOCTL interface sample

// 32 bit handle of the device
// set by OpenDevice(), and cleared by CloseDevice()
HANDLE ghDevice;

DWORD IoctlTest(void)
{
    DWORD BytesTransferred = 0; // actual number of bytes transferred
    CEUSB2_DRIVER_INFO DrvInfo;

    DWORD dwStatus = OpenDevice(); // open the device
    if(!CE_SUCCESS(dwStatus))
        return dwStatus; // error opening the device

    // call the driver
```

```

if(!DeviceIoControl(
    ghDevice,          // handle to device
    IOCTL_CEUSB2_GET_DRIVER_INFORMATION, // operation
    NULL, // input data buffer
    0, // size of input data buffer
    &DrvInfo, // output data buffer
    sizeof(CEUSB2_DRIVER_INFO), // size of output data buffer
    &BytesTransferred, // actual amount of data transferred
    NULL // overlapped information
))return GetLastError();

// display the version of this firmware
printf("Drvier name %s\nVersion : %u.%u\nBuild number %u\nBuild string %s\n",
    DrvInfo.Name, DrvInfo.MajorVersion, DrvInfo.MinorVersion,
    DrvInfo.BuildNumber, DrvInfo.BuildTypeStr);

//close the device
CloseDevice();
}

```

## 3 PROGRAMMING INTERFACE REFERENCE

### 3.1 I/O Control (IOCTL) Codes Reference

#### 3.1.1 IOCTL\_CEUSB2\_GET\_DRIVER\_INFORMATION

**Direction** Input

**Input buffer** None

**Input buffer length** 0

**Output buffer** CEUSB2\_DRIVER\_INFO structure object.

**Output buffer length** Size of CEUSB2\_DRIVER\_INFO structure.

**Bytes returned** Size of CEUSB2\_DRIVER\_INFO structure.

**Description** Retrieves versioning information from the driver as driver name, version (major and minor), interface version, build number and build type string (debug or release). See also CEUSB2\_DRIVER\_INFO.

#### 3.1.2 IOCTL\_CEUSB2\_DESCRIPTOR\_REQUEST

**Direction** Input/Output

**Input buffer** CEUSB2\_DESCRIPTOR\_INFO structure object.

**Input buffer length** Size of CEUSB2\_DESCRIPTOR\_INFO structure.

**Output buffer** Descriptor data buffer.

**Output buffer length** Descriptor data buffer size.

**Bytes returned** Actual number of descriptor bytes transferred

**Description** Retrieves or sets an USB descriptor (Set feature is not supported anymore). This IOCTL code is suitable to retrieve USB device, configuration and string descriptors of a CeUsb2 device although it can be used for the others. IOCTL\_CEUSB2\_GET\_INTERFACE\_DESCRIPTOR IOCTL code is used instead, to retrieve interface and endpoint descriptors.

For USB device descriptor output buffer is a pointer to device descriptor structure (USB\_DEVICE\_DESCRIPTOR) and output buffer size is the size of this structure.

For USB configuration descriptor output buffer is a pointer to configuration descriptor structure (USB\_CONFIGURATION\_DESCRIPTOR) and output buffer size is the size of this structure.

For USB string descriptor output buffer is the string descriptor structure (USB\_STRING\_DESCRIPTOR) at the requested index with the string concatenated to this structure, and output buffer size is its length together with the length of the string.

### 3.1.3 IOCTL\_CEUSB2\_GET\_INTERFACE\_DESCRIPTOR

**Direction** Input

**Input buffer** CEUSB2\_INTERFACE\_DESCRIPTOR\_INFO structure object

**Input buffer length** Size of CEUSB2\_INTERFACE\_DESCRIPTOR\_INFO structure

**Output buffer** Interface descriptor data buffer

**Output buffer length** Interface descriptor data buffer size

**Bytes returned** Actual number of interface descriptor bytes returned

**Description** Retrieves an USB interface descriptor from the driver.

### 3.1.4 IOCTL\_CEUSB2\_RESET

**Direction** Output

**Input buffer** None

**Input buffer length** 0

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Resets the USB port. Uses the IOCTL\_INTERNAL\_USB\_RESET\_PORT internal IOCTL code,

### 3.1.5 IOCTL\_CEUSB2\_RESET\_PIPE

**Direction** Output

**Input buffer** Unsigned long variable (pipe number).

**Input buffer length** Size of unsigned long type, sizeof(ULONG).

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Resets an USB Pipe.

### 3.1.6 IOCTL\_CEUSB2\_ABORT\_PIPE

**Direction** Output

**Input buffer** Unsigned long variable (pipe number).

**Input buffer length** Size of unsigned long type, sizeof(ULONG).

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Aborts an USB Pipe.

### 3.1.7 IOCTL\_CEUSB2\_FEATURE\_REQUEST

**Direction** Output

**Input buffer** CEUSB2\_FEATURE\_REQUEST\_INFO structure object.

**Input buffer length** Size of CEUSB2\_FEATURE\_REQUEST\_INFO structure.

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Sets or clears an USB feature on a device, interface, or endpoint. These features are device and firmware specific. Check the documentation of your firmware about the supported ones. See also CEUSB2\_FEATURE\_REQUEST\_INFO.

### 3.1.8 IOCTL\_CEUSB2\_VENDOR\_OR\_CLASS\_REQUEST

**Direction** Input/Output

**Input buffer** CEUSB2\_VENDOR\_OR\_CLASS\_REQUEST\_CONTROL structure object.

**Input buffer length** Size of CEUSB2\_VENDOR\_OR\_CLASS\_REQUEST\_CONTROL structure.

**Output buffer** Vendor or class request data buffer.

**Output buffer length** Vendor or class buffer size.

**Bytes returned** Actual amount of bytes transferred.

**Description** Performs a vendor or class specific USB request through CeUsb2 driver. Data buffer, data buffer length and bytes returned are vendor or class request specific. All vendor or class requests with their buffer organization are documented in the generic firmware document or firmware specific document.  
**Notes** For vendor requests whose recipient is the device itself, use VendorRequest function instead.

### 3.1.9 IOCTL\_CEUSB2\_VENDOR\_REQUEST

**Direction** Input/Output

**Input buffer** CEUSB2\_VENDOR\_REQUEST\_CONTROL structure object

**Input buffer length** Size of CEUSB2\_VENDOR\_REQUEST\_CONTROL structure.

**Output buffer** Vendor request data buffer.

**Output buffer length** Vendor request data buffer size.

**Bytes returned** Actual amount of bytes transferred.

**Description** Performs a vendor specific USB request through CeUsb2 driver. Data buffer, data buffer length and bytes returned are vendor request specific. All vendor requests with their buffer organization are documented in the generic firmware document or firmware specific document.

### 3.1.10 IOCTL\_CEUSB2\_GET\_LAST\_USBD\_ERROR

**Direction** Input

**Input buffer** None

**Input buffer length** 0

**Output buffer** Unsigned long variable (error code)

**Output buffer length** Size of unsigned long type, sizeof(ULONG)

**Bytes returned** Size of unsigned long type, sizeof(ULONG)

**Description** Retrieves the most recently occurred USB error code. Ceusb2 driver returns CEUSB2\_STATUS\_USBD\_ERROR status code if an USB specific error occurs, and stores another status code (USBD status code) in the driver which can be retrieved with this IOCTL code.

### 3.1.11 IOCTL\_CEUSB2\_GET\_CURRENT\_FRAME\_NUMBER

**Direction** Input

**Input buffer** None

**Input buffer length** 0

**Output buffer** Unsigned long variable (frame number).

**Output buffer length** Size of unsigned long type, sizeof(ULONG).

**Bytes returned** Size of unsigned long type, sizeof(ULONG).

**Description** Retrieves the current frame number on the USB bus.

### 3.1.12 IOCTL\_CEUSB2\_GET\_STATUS

**Direction** Input

**Input buffer** CEUSB2\_GET\_STATUS\_INFO structure object.

**Input buffer length** Size of CEUSB2\_GET\_STATUS\_INFO structure.

**Output buffer** Unsigned character variable (status).

**Output buffer length** Size of unsigned character type, sizeof(UCHAR).

**Bytes returned** Size of unsigned character type, sizeof(UCHAR).

**Description** Retrieves status from a CeUsb2 device, interface, endpoint, or other device-defined target. These statuses are device and firmware specific. Check the documentation of your firmware about the supported ones. See also CEUSB2\_GET\_STATUS\_INFO structure.

### 3.1.13 IOCTL\_CEUSB2\_SET\_DEVICE\_POWER\_STATE

**Direction** Output

**Input buffer** CEUSB2\_POWER\_INFO structure object.

**Input buffer length** Size of CEUSB2\_POWER\_INFO structure.

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Sets the power state of a CeUsb2 device. See also CEUSB2\_POWER\_INFO.

### 3.1.14 IOCTL\_CEUSB2\_GET\_DEVICE\_POWER\_STATE

**Direction** Output

**Input buffer** None

**Input buffer length** 0

**Output buffer** CEUSB2\_POWER\_INFO structure object.

**Output buffer length** Size of CEUSB2\_POWER\_INFO structure.

**Bytes returned** Size of CEUSB2\_POWER\_INFO structure.

**Description** Retrieves the current power state of a CeUsb2 device. See also CEUSB2\_POWER\_INFO.

### 3.1.15 IOCTL\_CEUSB2\_GET\_CONFIGURATION

**Direction** Input

**Input buffer** None

**Input buffer length** 0

**Output buffer** Unsigned character variable (configuration number)

**Output buffer length** Size of unsigned character type, sizeof(UCHAR)

**Bytes returned** Size of unsigned character type, sizeof(UCHAR)

**Description** Retrieves the current USB configuration number from a CeUsb2 device. Configuration number is indexed starting from 1. 0 means the device is in an unconfigured state.

### 3.1.16 IOCTL\_CEUSB2\_SELECT\_CONFIGURATION

**Direction** Output

**Input buffer** Unsigned character variable (configuration number)

**Input buffer length** Size of unsigned character type, sizeof(UCHAR)

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Selects the specified USB configuration for the device. Selecting 0 puts the device in an unconfigured state.

### 3.1.17 IOCTL\_CEUSB2\_GET\_INTERFACE\_INFORMATION

**Direction** Input

**Input buffer** None

**Input buffer length** 0

**Output buffer** USBD\_INTERFACE\_INFORMATION structure object concatenated with USBD\_PIPE\_INFORMATION objects if requested.

**Output buffer length** Size of USBD\_INTERFACE\_INFORMATION structure together with the size of USBD\_PIPE\_INFORMATION structure sizes, depending on the number of pipes.

**Bytes returned** Size of USBD\_INTERFACE\_INFORMATION structure together with the size of USBD\_PIPE\_INFORMATION structure sizes, depending on the number of pipes.

**Description** Retrieves information about the currently selected interface and alternate setting. This interface information may include the pipes information if requested.

To retrieve the pipes information also, two calls to this IOCTL code should be performed sequentially. First call will be with the buffer USBD\_INTERFACE\_INFORMATION structure alone and the size of this structure as output buffer size. The returned buffer includes the interface information together with the first pipe, if it has at least one. You also get the total number of the pipes. The second call's output buffer should be big enough to hold the whole interface information together with its pipes.

### 3.1.18 IOCTL\_CEUSB2\_GET\_INTERFACE

**Direction** Input

**Input buffer** Unsigned character variable (interface number).

**Input buffer length** Size of unsigned character type, sizeof(UCHAR).

**Output buffer** Unsigned character variable (alternate setting).

**Output buffer length** Size of unsigned character type, sizeof(UCHAR).

**Bytes returned** Size of unsigned character type, sizeof(UCHAR).

**Description** Retrieves the current USB alternate setting for an interface in the current configuration.

### 3.1.19 IOCTL\_CEUSB2\_SELECT\_INTERFACE

**Direction** Output

**Input buffer** CEUSB2\_INTERFACE\_INFO structure object concatenated with 32 bit unsigned long variables which specify the maximum transfer sizes of the pipes.

**Input buffer length** Size of CEUSB2\_INTERFACE\_INFO structure together with the size of the unsigned long data array (maximum transfer sizes of the pipes).

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Selects an USB interface and alternate setting for the current configuration. It can also change the maximum transfer sizes of the pipes for the selected interface.

### 3.1.20 IOCTL\_CEUSB2\_READ\_ISO

**Direction** Input

**Input buffer** CEUSB2\_ISO\_TRANSFER\_INFO structure object.

**Input buffer length** Size of CEUSB2\_ISO\_TRANSFER\_INFO structure.

**Output buffer** Data buffer.

**Output buffer length** Data buffer size.

**Bytes returned** Actual number of bytes read.

**Description** Reads specified amount of data from an isochronous pipe  
See also CEUSB2\_ISO\_TRANSFER\_INFO.

### 3.1.21 IOCTL\_CEUSB2\_WRITE\_ISO

**Direction** Output

**Input buffer** CEUSB2\_ISO\_TRANSFER\_INFO structure object

**Input buffer length** Size of CEUSB2\_ISO\_TRANSFER\_INFO structure

**Output buffer** Data buffer.

**Output buffer length** Data buffer size.

**Bytes returned** Actual number of bytes written.

**Description** Writes specified amount of data to an isochronous pipe.  
See also CEUSB2\_ISO\_TRANSFER\_INFO.

### 3.1.22 IOCTL\_CEUSB2\_START\_ISO\_STREAM

**Direction** Output

**Input buffer** CEUSB2\_ISO\_TRANSFER\_INFO\_EX structure object.

**Input buffer length** Size of CEUSB2\_ISO\_TRANSFER\_INFO\_EX structure.

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Starts internal isochronous data streaming, and allocates buffers for streaming FIFOs. See also CEUSB2\_ISO\_TRANSFER\_INFO\_EX structure.

### 3.1.23 IOCTL\_CEUSB2\_STOP\_ISO\_STREAM

**Direction** Output

**Input buffer** None

**Input buffer length** 0

**Output buffer** None

**Output buffer length** 0

**Bytes returned** 0

**Description** Stops internal isochronous data streaming, and deallocates buffers reserved for streaming FIFOs.

### 3.1.24 IOCTL\_CEUSB2\_READ\_ISO\_STREAM

**Direction** Input

**Input buffer** None

**Input buffer length** 0

**Output buffer** Data buffer

**Output buffer length** Data buffer length

**Bytes returned** 0

**Description** Reads specified amount of data from an isochronous stream.

### 3.1.25 IOCTL\_CEUSB2\_READ\_BULK

**Direction** Input

**Input buffer** Unsigned long variable (pipe number).

**Input buffer length** Size of unsigned long type, sizeof(ULONG).

**Output buffer** Data buffer.  
**Output buffer length** Data buffer size.  
**Bytes returned** Actual number of bytes read.  
**Description** Reads specified amount of data from a bulk pipe.

### 3.1.26 IOCTL\_CEUSB2\_WRITE\_BULK

**Direction** Output  
**Input buffer** Unsigned long variable (pipe number)  
**Input buffer length** Size of unsigned long type, sizeof(ULONG)  
**Output buffer** Data buffer.  
**Output buffer length** Data buffer size.  
**Bytes returned** Actual number of bytes written.  
**Description** Writes specified amount of data to a bulk pipe.

### 3.1.27 IOCTL\_CEUSB2\_READ\_CONTROL

**Direction** Input  
**Input buffer** CEUSB2\_CONTROL\_TRANSFER\_INFO structure object.  
**Input buffer length** Size of CEUSB2\_CONTROL\_TRANSFER\_INFO structure.  
**Output buffer** Data buffer.  
**Output buffer length** Data buffer size.  
**Bytes returned** Actual number of bytes read.  
**Description** Reads specified amount of data from a control pipe.

### 3.1.28 IOCTL\_CEUSB2\_WRITE\_CONTROL

**Direction** Output  
**Input buffer** CEUSB2\_CONTROL\_TRANSFER\_INFO structure object.  
**Input buffer length** Size of CEUSB2\_CONTROL\_TRANSFER\_INFO structure.  
**Output buffer** Data buffer.  
**Output buffer length** Data buffer size.  
**Bytes returned** Actual number of bytes written.  
**Description** Writes specified amount of data to a control pipe.

## 3.2 Interface Structures

### 3.2.1 USB\_DEVICE\_DESCRIPTOR

```
typedef struct _USB_DEVICE_DESCRIPTOR  
{
```

```

    UCHAR    bLength;
    UCHAR    bDescriptorType;
    USHORT   bcdUSB;
    UCHAR    bDeviceClass;
    UCHAR    bDeviceSubClass;
    UCHAR    bDeviceProtocol;
    UCHAR    bMaxPacketSize0;
    USHORT   idVendor;
    USHORT   idProduct;
    USHORT   bcdDevice;
    UCHAR    iManufacturer;
    UCHAR    iProduct;
    UCHAR    iSerialNumber;
    UCHAR    bNumConfigurations;
} USB_DEVICE_DESCRIPTOR, *PUSB_DEVICE_DESCRIPTOR;

```

bLength - Specifies the length, in bytes, of this device descriptor.

bDescriptorType - Set to USB\_DEVICE\_DESCRIPTOR\_TYPE.

bcdUSB - Binary-coded decimal number which identifies the version of the USB specification that this descriptor structure complies with.

bDeviceClass - Class code of the device as assigned by the USB specification group.

bDeviceSubClass - Subclass code of the device as assigned by the USB specification group.

bDeviceProtocol - Protocol code of the device as assigned by the USB specification group.

bMaxPacketSize0 - Specifies the maximum packet size, in bytes, for endpoint zero of the device. The value is set to 8, 16, 32, or 64.

idVendor - Vendor identifier for the device as assigned by the USB specification committee.

idProduct - Product identifier assigned by Cesium.

bcdDevice - Binary-coded decimal number which identifies the version of the device.

iManufacturer - Specifies a device-defined index of the string descriptor that provides a string containing the name of the manufacturer of this device.

iProduct - Specifies a device-defined index of the string descriptor that provides a string that contains a description of the device.

iSerialNumber - Specifies a device-defined index of the string descriptor that provides a string that contains a manufacturer-determined serial number for the device.

bNumConfigurations - Total number of possible configurations for the device.

### 3.2.2 USB\_CONFIGURATION\_DESCRIPTOR

```

typedef struct _USB_CONFIGURATION_DESCRIPTOR
{
    UCHAR    bLength;
    UCHAR    bDescriptorType;
    USHORT   wTotalLength;

```

```

    UCHAR    bNumInterfaces;
    UCHAR    bConfigurationValue;
    UCHAR    iConfiguration;
    UCHAR    bmAttributes;
    UCHAR    MaxPower;
} USB_CONFIGURATION_DESCRIPTOR, *PUSB_CONFIGURATION_DESCRIPTOR;

```

**Description :** USB configuration descriptor structure.

**Members :**

bLength - Specifies the length, in bytes, of this structure.

bDescriptorType - Set to USB\_CONFIGURATION\_DESCRIPTOR\_TYPE.

wTotalLength - Specifies the total length, in bytes, of all data for the configuration. The length includes all interface, endpoint, class, or vendor-specific descriptors returned with the configuration descriptor.

bNumInterfaces - Total number of interfaces supported by this configuration.

bConfigurationValue - Index that identifies this USB configuration. This value starts from 1 (default configuration). 0 indicates unconfigured state.

iConfiguration - Optional device defined index of the string descriptor for this configuration.

bmAttributes - This one byte member describes the behavior of this configuration. Table 3.1 shows the bit-map for bmAttributes.

Bit	Description
0 – 4	Reserved.
5	Set to 1, if this configuration supports USB remote wakeup feature.
6	Set to 1, if this configuration is self-powered and does not use power from the USB bus.
7	Set to 1, if this configuration is powered by the bus.

*Table 3.1 – bmAttributes bit-map.*

If bmAttributes bits six and seven are both set, then the device is powered both by the bus and a source external to the bus.

MaxPower - Specifies the power requirements of this device in two mA units. This field is valid only if bit seven is set in bmAttributes.

### 3.2.3 USB\_INTERFACE\_DESCRIPTOR

```

typedef struct _USB_INTERFACE_DESCRIPTOR
{
    UCHAR    bLength;
    UCHAR    bDescriptorType;
    UCHAR    bInterfaceNumber;
    UCHAR    bAlternateSetting;
    UCHAR    bNumEndpoints;

```

```

    UCHAR    bInterfaceClass;
    UCHAR    bInterfaceSubClass;
    UCHAR    bInterfaceProtocol;
    UCHAR    iInterface;
} USB_INTERFACE_DESCRIPTOR, *PUSB_INTERFACE_DESCRIPTOR;

```

**Description :** USB interface descriptor structure.

**Members :**

bLength - Specifies the length, in bytes, of this interface descriptor.

bDescriptorType - Specifies the descriptor type. It is set to USB\_INTERFACE\_DESCRIPTOR\_TYPE.

bInterfaceNumber - Index number of this interface.

bAlternateSetting - Index number of this alternate setting of the interface.

bNumEndpoints – Total number of endpoints that are used by the interface, excluding the default control endpoint (endpoint 0).

bInterfaceClass - Class code of the device as assigned by the USB specification group.

bInterfaceSubClass - Subclass code of the device as assigned by the USB specification group.

bInterfaceProtocol - Protocol code of the device as assigned by the USB specification group.

iInterface - Index of a string descriptor that describes the interface.

### 3.2.4 USB\_ENDPOINT\_DESCRIPTOR

```

typedef struct _USB_ENDPOINT_DESCRIPTOR
{
    UCHAR    bLength;
    UCHAR    bDescriptorType;
    UCHAR    bEndpointAddress;
    UCHAR    bmAttributes;
    USHORT   wMaxPacketSize;
    UCHAR    bInterval;
} USB_ENDPOINT_DESCRIPTOR, *PUSB_ENDPOINT_DESCRIPTOR;

```

**Description :** USB endpoint descriptor structure.

**Members :**

bLength - Specifies the length, in bytes, of this endpoint descriptor.

bDescriptorType – Set to USB\_ENDPOINT\_DESCRIPTOR\_TYPE.

bEndpointAddress - Specifies the USB defined endpoint address. The four low-order bits specify the endpoint number. The high-order bit specifies the direction of data flow on this endpoint: 1 for in, 0 for out.

bmAttributes - The two low-order bits specify the endpoint type. Possible values are:

- 0 – Control Endpoint
- 1 – Isochronous Endpoint
- 2 – Bulk Endpoint

### 3 – Interrupt Endpoint.

The upper 6 bits of this field is not used and reserved for future use.

wMaxPacketSize - Specifies the maximum packet size that this endpoint supports.

bInterval - For interrupt and isochronous endpoints, bInterval specifies the polling interval of USB data transfer frames (CeUsb2 doesn't support interrupt endpoints).

#### 3.2.5 USB\_STRING\_DESCRIPTOR

```
typedef struct _USB_STRING_DESCRIPTOR
{
    UCHAR        bLength;
    UCHAR        bDescriptorType;
    WCHAR        bString[1];
} USB_STRING_DESCRIPTOR, *PUSB_STRING_DESCRIPTOR;
```

**Description :** USB string descriptor structure.

**Members :**

bLength - Specifies the length, in bytes, of the descriptor.

bDescriptorType – Set to USB\_STRING\_DESCRIPTOR\_TYPE.

bString - Points to a user allocated buffer that contains a unicode string with the requested string descriptor.

#### 3.2.6 USB\_COMMON\_DESCRIPTOR

```
typedef struct _USB_COMMON_DESCRIPTOR {
    UCHAR        bLength;
    UCHAR        bDescriptorType;
} USB_COMMON_DESCRIPTOR, *PUSB_COMMON_DESCRIPTOR;
```

**Description :** USB common descriptor structure.

**Members :**

bLength - Specifies the length, in bytes, of the descriptor.

bDescriptorType - Specifies the descriptor type.

#### 3.2.7 USB\_INTERFACE\_INFORMATION

```
typedef struct _USB_INTERFACE_INFORMATION
{
    USHORT        Length;
```

```

    UCHAR    InterfaceNumber;
    UCHAR    AlternateSetting;
    UCHAR    Class;
    UCHAR    SubClass;
    UCHAR    Protocol;
    UCHAR    Reserved;
    PVOID    InterfaceHandle;
    ULONG    NumberOfPipes;
    USBD_PIPE_INFORMATION Pipes[1];
} USBD_INTERFACE_INFORMATION, *PUSB_INTERFACE_INFORMATION;

```

**Description :** USB interface information structure.

**Members :**

Length - Specifies the length, in bytes, of this structure.

InterfaceNumber - Device defined index identifier for this interface.

AlternateSetting - Device-defined index identifier that indicates which alternate setting this interface is using or should use.

Class - USB-assigned identifier to specify a USB-defined class that this interface conforms to.

SubClass – USB assigned identifier to specify a USB-defined subclass that this interface conforms to. This code is specific to the code in Class.

Protocol – USB assigned identifier to specify a USB-defined protocol that this interface conforms to. This code is specific to the codes in Class and SubClass.

InterfaceHandle - Host controller driver-defined handle that is used to access this interface.

NumberOfPipes - Specifies the number of pipes (endpoints) in this interface.

Pipes - Variable length array of USBD\_PIPE\_INFORMATION structures to describe each pipe in the interface.

### 3.2.8 USBD\_PIPE\_INFORMATION

```

typedef struct _USB_PIPE_INFORMATION
{
    USHORT    MaximumPacketSize;
    UCHAR    EndpointAddress;
    UCHAR    Interval;
    USBD_PIPE_TYPE PipeType;
    PVOID    PipeHandle;
    ULONG    MaximumTransferSize;
    ULONG    PipeFlags;
} USB_PIPE_INFORMATION, *PUSB_PIPE_INFORMATION;

```

**Description :** USB pipe information structure.

**Members :**

MaximumPacketSize - Maximum packet size, in bytes, that this pipe supports.

EndpointAddress - Bus address for this pipe. The four low-order bits specify the endpoint number. The high-order bit specifies the direction of data flow on this pipe: 1 for in, 0 for out.

Interval - For interrupt and isochronous pipes, bInterval specifies the polling interval of USB data transfer frames (CeUsb2 doesn't support interrupt pipes).

PipeType - Specifies what type of transfers this pipe uses. See also USB\_D\_PIPE\_TYPE enumeration type.

PipeHandle - Specifies a host controller driver defined handle that is used to access this pipe.

MaximumTransferSize - Specifies the maximum size, in bytes, for a transfer request on this pipe.

### 3.2.9 CEUSB2\_DRIVER\_INFO

```
typedef struct _CEUSB2_DRIVER_INFO
{
    CHAR        Name[32];
    USHORT     InterfaceVersion;
    UCHAR      MajorVersion;
    UCHAR      MinorVersion;
    ULONG      BuildNumber;
    CHAR       BuildTypeStr[12];
}CEUSB2_DRIVER_INFO,*PCEUSB2_DRIVER_INFO;
```

**Description :** Driver information structure.

**Members :**

Name – 32 byte driver executable name.

InterfaceVersion - Driver interface version (programming interface).

MajorVersion - Driver major version number.

MinorVersion - Driver minor version number.

BuildNumber - Driver build number.

BuildTypeStr – 12 byte driver build type string (i.e. "debug", "release").

### 3.2.10 CEUSB2\_DESCRIPTOR\_INFO

```
typedef struct _CEUSB2_DESCRIPTOR_INFO
{
    CEUSB2_REQUEST_RECIPIENT Recipient;
    UCHAR      Direction;
    UCHAR      DescriptorType;
    UCHAR      Index;
    USHORT     LanguageId;
}CEUSB2_DESCRIPTOR_INFO,*PCEUSB2_DESCRIPTOR_INFO;
```

**Description :** Descriptor information structure which is used to perform descriptor requests through CeUsb2 driver.

**Members :**

Recipient – Descriptor request recipient (Not used anymore, recipient is always the device itself).

Direction - Descriptor request direction (0 - OUT , 1 – IN). This field is not used anymore, descriptor requests are always IN.

DescriptorType - Descriptor type, defined in UsbDef.h file.

Index - Index value used by some descriptor requests.

LanguageId - Language id used with string descriptors.

### 3.2.11 CEUSB2\_INTERFACE\_DESCRIPTOR\_INFO

```
typedef struct _CEUSB2_INTERFACE_DESCRIPTOR_INFO
{
    UCHAR        ConfigNo;
    UCHAR        InterfaceNumber;
    UCHAR        AlternateSetting;
}CEUSB2_INTERFACE_DESCRIPTOR_INFO,*PCEUSB2_INTERFACE_DESCRIPTOR_INFO;
```

**Description :** USB interface descriptor information structure.

**Members :**

ConfigNo - Configuration number

InterfaceNumber - Interface number

AlternateSetting - Alternate setting

### 3.2.12 CEUSB2\_INTERFACE\_INFO

```
typedef struct _CEUSB2_INTERFACE_INFO
{
    UCHAR        InterfaceNumber;
    UCHAR        AlternateSetting;
    ULONG        MaxTransferSize[1];
}CEUSB2_INTERFACE_INFO,*PCEUSB2_INTERFACE_INFO;
```

**Description :** Information structure for the current USB interface.

**Members :**

InterfaceNumber - Interface number.

AlternateSetting - Alternate setting.

MaxTransferSize – The first element of an array of 32 bit unsigned long variables which specify the maximum transfer sizes of the pipes. The remaining elements can be concatenated to this structure.

### 3.2.13 CEUSB2\_VENDOR\_OR\_CLASS\_REQUEST\_CONTROL

```
typedef struct _CEUSB2_VENDOR_OR_CLASS_REQUEST_CONTROL
{
    CEUSB2_REQUEST_RECIPIENT Recipient;
    UCHAR        RequestType;
    UCHAR        Request;
    UCHAR        Direction;
```

```

        USHORT      Value;
        USHORT      Index;
}CEUSB2_VENDOR_OR_CLASS_REQUEST_CONTROL,
*PCEUSB2_VENDOR_OR_CLASS_REQUEST_CONTROL;

```

**Description :** Vendor or class request control structure.

**Members :**

Recipient – Vendor or class request recipient

RequestType – Vendor or class request type (0 – class, 1 – vendor).

Request – Vendor or class request code.

Direction – Vendor or class request direction (0 - OUT, 1 – IN).

Value – Vendor or class request specific value.

Index – Vendor or class request specific index.

### 3.2.14 CEUSB2\_VENDOR\_REQUEST\_CONTROL

```

typedef struct _CEUSB2_VENDOR_REQUEST_CONTROL
{
    UCHAR      Request;
    UCHAR      Direction;
    USHORT     Value;
    USHORT     Index;
}CEUSB2_VENDOR_REQUEST_CONTROL, *PCEUSB2_VENDOR_REQUEST_CONTROL;

```

**Description :** Vendor request control structure.

**Members :**

Request – Vendor request code.

Direction – Vendor request direction (0 - OUT, 1 – IN).

Value – Vendor request specific value.

Index – Vendor request specific index.

### 3.2.15 CEUSB2\_GET\_STATUS\_INFO

```

typedef struct _CEUSB2_GET_STATUS_INFO
{
    CEUSB2_REQUEST_RECIPIENT Recipient;
    USHORT      Index;
}CEUSB2_GET_STATUS_INFO, *PCEUSB2_GET_STATUS_INFO;

```

**Description :** USB get status request control structure.

**Members :**

Recipient - Request recipient.

Index - Index value used with get status request (firmware specific).

### 3.2.16 CEUSB2\_FEATURE\_REQUEST\_INFO

```

typedef struct _CEUSB2_FEATURE_REQUEST_INFO
{
    CEUSB2_REQUEST_RECIPIENT Recipient;
    UCHAR          SetClear;
    USHORT         FeatureSelector;
    USHORT         Index;
}CEUSB2_FEATURE_REQUEST_INFO, *PCEUSB2_FEATURE_REQUEST_INFO;

```

**Description :** USB feature request control structure.

**Members :**

Recipient – Feature request recipient

SetClear - Feature request type (0 - clear, 1 – set).

FeatureSelector - Feature code (firmware specific).

Index - Index value used with feature request (firmware specific).

### 3.2.17 CEUSB2\_POWER\_INFO

```

typedef struct _CEUSB2_POWER_INFO
{
    CEUSB2_POWER_STATE PowerState;
    UCHAR          Wait;
}CEUSB2_POWER_INFO, *PCEUSB2_POWER_INFO;

```

**Description :** Get/Set power request control information structure.

**Members :**

PowerState - Power state to set or retrieved.

Wait – If 1, driver waits for the completion of the request, otherwise returns immediately. This field is valid for only for set power state request.

### 3.2.18 CEUSB2\_ISO\_TRANSFER\_INFO

```

typedef struct _CEUSB2_ISO_TRANSFER_INFO
{
    ULONG          PipeNumber;
    ULONG          PacketSize;
    ULONG          FramesPerBuffer;
    ULONG          BufferCount;
}CEUSB2_ISO_TRANSFER_INFO, *PCEUSB2_ISO_TRANSFER_INFO;

```

**Description :** Isochronous transfer information structure.

**Members :**

PipeNumber - Pipe number.

PacketSize - Amount of ISO data to read during each frame. This value usually corresponds to the max packet size of the ISO endpoint, but can be less.

FramesPerBuffer - number of USB frames of data to transfer in a single URB (USB Request Block).

BufferCount - Number of transfer URBs to use for this transfer.

### 3.2.19 CEUSB2\_ISO\_TRANSFER\_INFO\_EX

```
typedef struct _CEUSB2_ISO_TRANSFER_INFO_EX
{
    ULONG      PipeNumber;
    ULONG      PacketSize;
    ULONG      FramesPerBuffer;
    ULONG      BufferCount;
    ULONG      StreamBufferSize;
}CEUSB2_ISO_TRANSFER_INFO_EX, *PCEUSB2_ISO_TRANSFER_INFO_EX;
```

**Description :** Isochronous streaming information structure.

**Members :**

PipeNumber - Pipe number.

PacketSize - Amount of ISO data to read during each frame. This value usually corresponds to the max packet size of the ISO endpoint, but can be less.

FramesPerBuffer - number of USB frames of data to transfer in a single URB (USB Request Block).

BufferCount - Number of transfer URBs to use for this transfer.

StreamBufferSize – Internal stream FIFO buffer size (in bytes).

### 3.2.20 CEUSB2\_CONTROL\_TRANSFER\_INFO

```
typedef struct _CEUSB2_CONTROL_TRANSFER_INFO
{
    UCHAR      SetupData[8];
    ULONG      PipeNumber;
}CEUSB2_CONTROL_TRANSFER_INFO, *PCEUSB2_CONTROL_TRANSFER_INFO;
```

**Description :** Control transfer information structure.

**Members :**

SetupData[8] - 8 bit setup data used with all control transfers.

PipeNumber - Pipe number (Use 0).

## 3.3 Interface Enumeration Types

### 3.3.1 USBD\_PIPE\_TYPE

```
typedef enum _USB_D_PIPE_TYPE
{
    _UsbdPipeTypeControl,
    _UsbdPipeTypeIsochronous,
    _UsbdPipeTypeBulk,
}
```

```
_UsbdPipeTypeInterrupt  
} USBD_PIPE_TYPE;
```

**Description :** Specifies the pipe type.

**Elements:**

\_UsbdPipeTypeControl – Control pipe.  
\_UsbdPipeTypeIsochronous – Isochronous pipe.  
\_UsbdPipeTypeBulk – Bulk pipe.  
\_UsbdPipeTypeInterrupt – Interrupt pipe.

### 3.3.2 CEUSB2\_REQUEST\_RECIPIENT

```
enum CEUSB2_REQUEST_RECIPIENT  
{  
    RECIPIENT_DEVICE = 0,  
    RECIPIENT_INTERFACE,  
    RECIPIENT_ENDPOINT,  
    RECIPIENT_OTHER  
};
```

**Description :** Specifies the recipient of a request.

**Elements:**

RECIPIENT\_DEVICE – Recipient is device.  
RECIPIENT\_INTERFACE – Recipient is an interface.  
RECIPIENT\_ENDPOINT – Recipient is an endpoint.  
RECIPIENT\_OTHER – Recipient is another device defined target.

### 3.3.3 CEUSB2\_POWER\_STATE

```
enum CEUSB2_POWER_STATE  
{  
    _PowerDeviceUnspecified = 0,  
    _PowerDeviceD0,  
    _PowerDeviceD1,  
    _PowerDeviceD2,  
    _PowerDeviceD3,  
    _PowerDeviceMaximum  
};
```

**Description :** Specifies the power states of a CeUsb2 device.

**Elements:**

\_PowerDeviceUnspecified - Power state is undefined.  
\_PowerDeviceD0 – Device power is fully ON.  
\_PowerDeviceD1 – Device power is almost ON.  
\_PowerDeviceD2 – Device power is almost OFF.  
\_PowerDeviceD3 – Device power is fully OFF.  
\_PowerDeviceMaximum – Power is maximum.

