the high-performance USB foundation software

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259
http://www.expresslogic.com

Part Number: 000-1010

Revision 5.0

Contents

About This Guide

This guide contains comprehensive information about USBX™, the high-performance USB foundation software from Express Logic, Inc.

It is intended for the embedded real-time software developer. The developer should be familiar with standard real-time operating system functions, the USB specification, and the C programming language.

For technical information related to USB, see the USB specification and USB Class specifications that can be downloaded at

http://www.USB.org/developers

Organization

Chapter 1 Contains an introduction to USBX.

Chapter 2 Gives the basic steps to install and use USBX with your ThreadX application.

Chapter 3 Provides an overview of the USBX Host Stack and describes its components.

Chapter 4 Details the application's interface to USBX in host mode.

Chapter 5 Details the application interface to USBX in device mode.

Chapter 6	Provides a reference to each USBX device service.
Appendix A	Contains supplementary information about the USB device and DPUMP classes.
Appendix B	Lists the USBX constants
Appendix C	Lists the USBX data types
Appendix D	Lists the primary language and sublanguage identifiers
Index	Gives page references to the guide's primary subjects.

Guide Conventions

Italics	Typeface denotes book titles, emphasizes important words, and indicates variables.
Boldface	Typeface denotes file names, key words, and further emphasizes important words and variables.

USBX Data Types

In addition to the custom USBX control structure data types, there is a series of special data types that are used in USBX service call interfaces. These special data types map directly to data types of the underlying C compiler. This is done to insure portability between different C compilers. The exact implementation is inherited from ThreadX and can be found in the **tx_port.h** file included in the ThreadX distribution.

The following is a list of USBX service call data types and their associated meanings:

UINT	Basic unsigned integer. This type must support 8-bit unsigned data; however, it is mapped to the most convenient unsigned data type.
ULONG	Unsigned long type. This type must support 32-bit unsigned data.
VOID	Almost always equivalent to the compiler's void type.
CHAR	Most often a standard 8-bit character type.

Additional data types are used within the USBX source. They are located in either the **tx_port.h** or **ux_port.h** files.

Customer Support Center

Support engineers	858.613.6640
Support fax	858.521.4259
Support email	support@expresslogic.com
Web page	http://www.expresslogic.com

Latest Product Information

Visit the Express Logic web site and select the "Support" menu option to find the latest support

information, including information about the latest USBX product releases.

What We Need From You

To more efficiently resolve your support request, provide us with the following information in your email request:

- A detailed description of the problem, including frequency of occurrence and whether it can be reliably reproduced.
- A detailed description of any changes to the application and/or USBX that preceded the problem.
- The contents of the ***_tx_version_id*** and ***_ux_version_id*** strings found in the **tx_port.h** and **ux_port.h** files of your distribution. These strings will provide us valuable information regarding your run-time environment.
- The type of USBX host and/or device controllers installed in the system.
- The type of USBX host and/or device classes installed in the system

Where to Send Comments About This Guide

The staff at Express Logic is always striving to provide you with better products. To help us achieve this goal, email any comments and suggestions to the Customer Support Center at

support@expresslogic.com

Please enter “USBX User Guide” in the subject line.

CHAPTER 1

Introduction to USBX

USBX is a full-featured USB stack for deeply embedded applications. This chapter introduces USBX, describing its applications and benefits:

USBX Features

USBX supports the two existing USB specifications: 1.1 and 2.0. It is designed to be scalable and will accommodate simple USB topologies with only one connected device as well as complex topologies with multiple devices and cascading hubs. USBX supports all the data transfer types of the USB protocols: control, bulk, interrupt, and isochronous.

USBX supports both the host side and the device side. Each side is comprised of three layers:

- Controller layer
- Stack layer
- Class layer

The relationship among the USB layers is illustrated in the following figure:

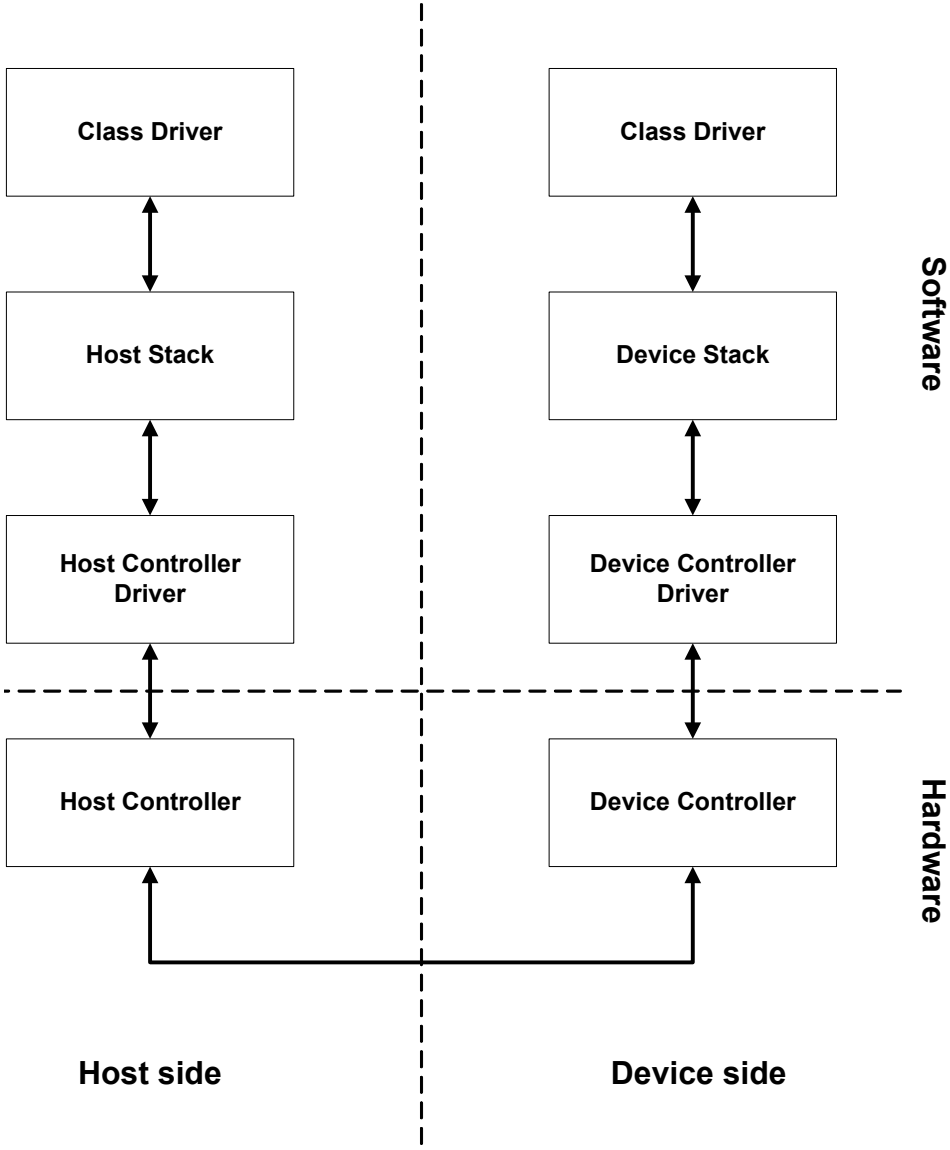


FIGURE 1. Relationship among USB Layers

Product Highlights

- Complete ThreadX processor support
- No royalties
- Complete ANSI C source code
- Real-time performance
- Responsive technical support
- Multiple host controller support
- Multiple class support
- Multiple class instances
- Integration of classes with ThreadX and FileX
- Support for USB devices with multiple configuration
- Support for USB composite devices
- Support for cascading hubs
- Support for USB power management

Powerful Services of USBX

Multiple Host Controller Support

USBX can support multiple USB host controllers running concurrently. This feature allows USBX to support the USB 2.0 standard using the backward compatibility scheme associated with most USB 2.0 host controllers on the market today.

USB Software Scheduler

USBX contains a USB software scheduler necessary to support USB controllers that do not have hardware list processing. The USBX software scheduler will organize USB transfers with the correct frequency of service and priority, and instruct the USB controller to execute each transfer.

Complete USB Device Framework Support

USBX can support the most demanding USB devices, with multiple configurations, multiple interfaces, and multiple alternate settings.

Easy-to-use APIs

USBX provides the very best deeply embedded USB stack in a manner that is easy to understand and use. The USBX API makes the services intuitive and consistent. By using the provided USBX class APIs, the user application does not need to understand the complexity of the USB protocols.

U S B X

CHAPTER 2

Installation and Use of USBX

This chapter contains an introduction to USBX and a description of installation conditions, procedures, and use, including the following:

Host Considerations

Computer Type

Embedded development is usually performed on either IBM-PCs or Unix host computers. After the application is compiled, linked, and located on the host, it is downloaded to the target hardware for execution.

Download Interfaces

Usually the target download is conducted using an RS-232 serial interface; however, parallel interfaces, USB, and Ethernet are becoming more popular. See the development tool documentation for available options.

Debugging Tools

Debugging is done typically over the same link as the program image download. A variety of debuggers exist, ranging from small monitor programs running on the target through Background Debug Monitor (BDM) and In-Circuit Emulator (ICE) tools. Of course, the ICE tool provides the most robust debugging of actual target hardware.

Required Hard Disk Space

The source code for USBX is delivered in ASCII format and requires approximately 500 KBytes of space on the host computer's hard disk. Review the supplied **readme_usbx.txt** file for additional host system considerations and options.

Target Considerations

USBX requires between 24 KBytes and 64 KBytes of Read Only Memory (ROM) on the target machine. The amount of memory required is dependant on the type of controller used and the USB classes linked to USBX. Another 32 Kbytes of the target's Random

Access Memory (RAM) are required for USBX global data structures and memory pool. This memory pool can also be adjusted depending on the expected number of devices on the USB and the type of USB controller.The USBX device side requires roughly 10-12K of ROM depending on the type of device controller. The RAM memory usage depends on the type of class emulated by the device.

USBX also relies on ThreadX semaphores, mutexes, and threads for multiple thread protection, I/O suspension, and periodic processing for monitoring the USB bus topology.

Product Distribution

Two USBX packages are available—standard and premium. The standard package includes minimal source code, while the premium package contains the complete USBX source code. Either package is shipped on a single CD.

The contents of the distribution CD depends on the target processor, development tools, and the USBX package. Following is a list of the important files common to most product distributions:

readme_usbx.txt Contains specific information about the USBX port, including information about the target processor and the development tools.

ux_api.h C header file that contains all system equates, data structures, and service prototypes.

ux_port.h	C header file that contains all development-tool-specific data definitions and structures.
ux.lib	Binary version of the USBX C library. It is distributed with the standard package.
demo_usbx.c	C file containing a simple USBX demo.

All filenames are in lower-case. This naming convention makes it easier to convert the commands to Unix development platforms.

USBX Installation

Installation of USBX is straightforward. The following general instructions apply to virtually any installation. However, the **readme_usbx.txt** file should be examined for changes specific to the actual development tool environment.

Step 1: Backup the USBX distribution disk and store it in a safe location.

Step 2: Use the same directory in which you previously installed ThreadX on the host hard drive. All USBX names are unique and will not interfere with the previous USBX installation.

Step 3: Add a call to **ux_system_initialize** at or near the beginning of **tx_application_define.** This is where the USBX resources are initialized.

Step 4: Add a call to **ux_host_stack_initialize** and/or **ux_device_stack_initialize.**

Step 5: Add one or more calls to initialize the required USBX classes (either host and/or devices classes).

Step 6: Add one or more calls to initialize the host and/or device controllers available in the system.

Step 7: It may be required to modify the tx_ill file to add low-level hardware initialization and interrupt vector routing. This is specific to the hardware platform and will not be discussed here.

Step 8: Compile application source code and link with the USBX and ThreadX runtime libraries (FileX may also be required if the USB storage class is to be compiled), **usbx.a** (or **usbx.lib**) and **tx.a** (or **tx.lib**). The resulting can be downloaded to the target and executed!

Configuration Options

There are several configuration options for building the USBX library. All options are located in the **ux_port.h** file. The list below details each configuration option. Additional development tool options are described in the **readme_usbx.txt** file supplied on the distribution disk.

UX_PERIODIC_RATE

This value represents how many ticks per seconds for a specific hardware platform. The default is 1000 indicating 1 tick per millisecond.

UX_MAX_CLASSES

This value is the maximum number of classes that can be loaded by USBX. This value represents the class container and not the number of instances of a class. For instance, if a particular implementation of USBX needs the hub class, the

printer class, and the storage class, then the UX_MAX_CLASSES value can be set to 3 regardless of the number of devices that belong to these classes.

UX_MAX_SLAVE_CLASSES

This value is the maximum number of classes in the device stack that can be loaded by USBX.

UX_MAX_HCD This value represents the number of different host controllers available in the system. For USB 1.1 support, this value will usually be 1. For USB 2.0 support, this value can be more than 1. This value represents the number of concurrent host controllers running at the same time. If for instance there are two instances of OHCI running, or one EHCI and one OHCI controller running, the UX_MAX_HCD should be set to 2.

UX_MAX_DEVICES

This value represents the maximum number of devices that can be attached to the USB. Normally, the theoretical maximum number on a single USB is 127 devices. This value can be scaled down to conserve memory. Note that this value represents the total number of devices regardless of the

number of USB buses in the system.

UX_MAX_SLAVE_LUN
This value represents the current number of SCSI logical units represented in the device storage class driver.

UX_MAX_HOST_LUN
This value represents the maximum number of SCSI logical units represented in the host storage class driver

UX_SLAVE_REQUEST_CONTROL_MAX_LENGTH
This value represents the maximum number of bytes received on a control endpoint in the device stack. The default is 256 bytes but can be reduced in memory constraint environments

UX_SLAVE_REQUEST_DATA_MAX_LENGTH
This value represents the maximum number of bytes received on a bulk endpoint in the device stack. The default is 4096 bytes but can be reduced in memory constraint environments.

Source Code Tree

The USBX files are provided in several directories as illustrated in the following figure:

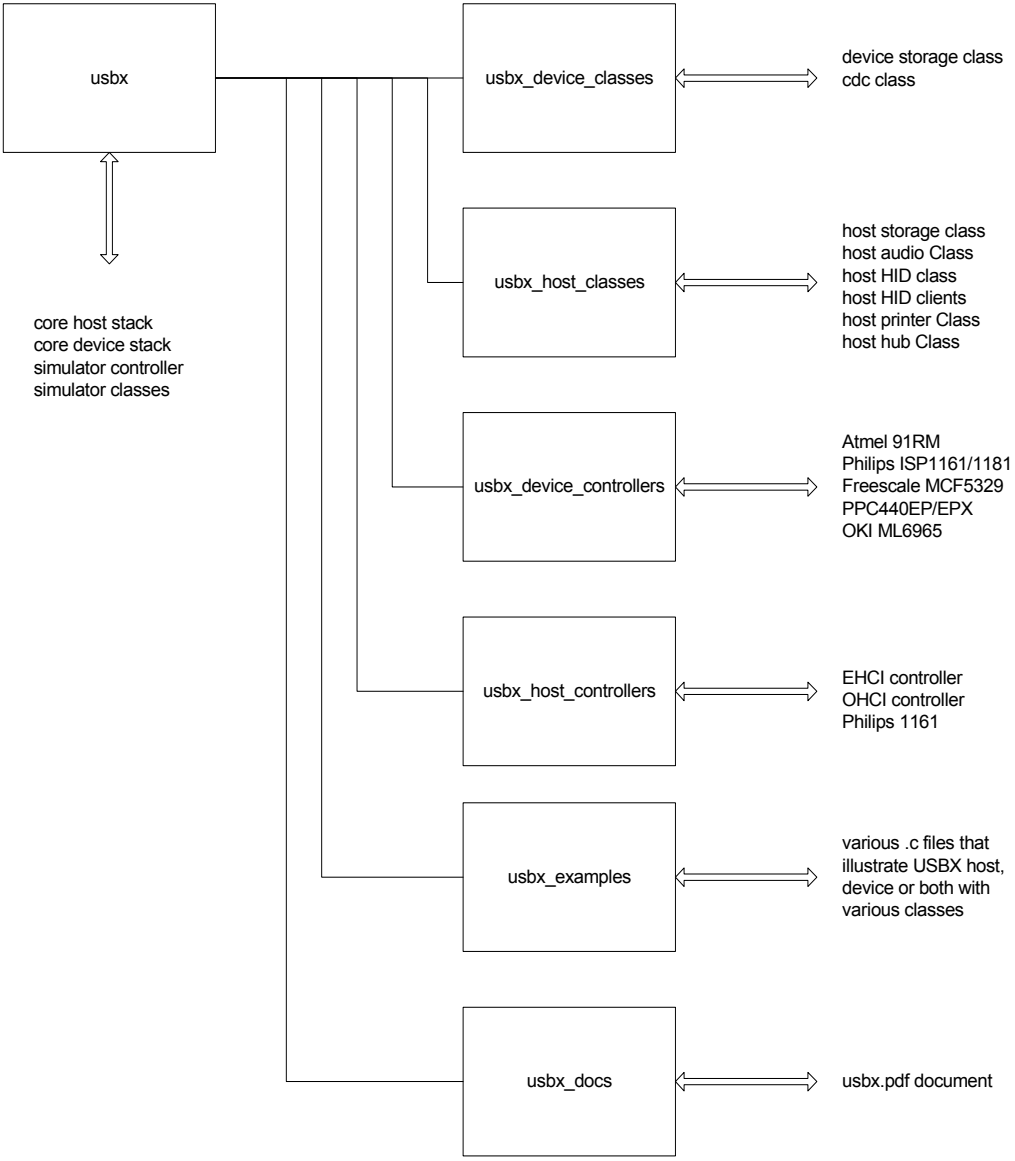


FIGURE 2. Source Code Tree

To make the files recognizable by their names, the following convention has been adopted:

File Suffix Name	File Content
ux_host_class_audio	USBX Audio Class files
ux_host_class_storage	USBX Storage Class files
ux_host_class_hid	USBX HID Class files
ux_host_class_hub	USBX Hub Class files
ux_host_class_printer	USBX Printer Class files
ux_dpump	USBX Host Dpump Class files
ux_host_stack	USBX Host Stack files
ux_hcd	USBX System Component
ux_device_stack	USBX Device Stack files
ux_device_class_storage	USBX Device Storage Class files
ux_device_class_dpump	USBX Device Storage Dpump files
ux_dcd	USBX Device Controller Driver files
ux_utility	USBX Utility System Component

Initialization of USBX resources

USBX has its own memory manager. The memory needs to be allocated to USBX before the host or device side of USBX is initialized. The USBX memory manager can accommodate systems in which memory can be cached.

The following function initializes USBX memory resources with 128K of regular memory and no separate pool for cache safe memory:

```
/* Initialize UsbX Memory */
   ux_system_initialize(memory_pointer,(128*1024),UX_NULL,0);
```

The prototype for the **ux_system_initialize** is as follows:

```
UINT  _ux_system_initialize(VOID *regular_memory_pool_start,
                            ULONG regular_memory_size,
                            VOID *cache_safe_memory_pool_start,
                            ULONG cache safe memory size)
```

Input parameters are as follows:

```
VOID *regular_memory_pool_start     : Beginning of the regular memory pool
ULONG regular_memory_size           : Size of the regular memory pool
VOID *cache_safe_memory_pool_start  : Beginning of the cache safe memory pool
ULONG cache_safe_memory_size        : Size of the cache safe memory pool
```

Not all systems require the definition of cache safe memory. In such a system, the values passed during the initialization for the memory pointer will be set to **UX_NULL** and the size of the pool to 0. USBX will then use the regular memory pool in lieu of the cache safe pool.

In a system in which the regular memory is not cache safe and a controller needs to perform DMA memory access (such as OHCI and EHCI controllers, among others), it is necessary to define a memory pool in a cache safe zone.

Definition of USB Host Controllers

At least one USB host controller must be defined for USBX to operate in host mode. The application initialization file should contain this definition. The example below refers to the OHCI USB controller. For other controllers, like EHCI, the name of the controller and function entry definition has to be changed accordingly.

The following line performs the definition of a OHCI controller:

```
ux_host_stack_hcd_register("ux_hcd_ohci", ux_hcd_ohci_initialize, 0xd0000, 0x0a);
```

The **ux_host_stack_hcd_register** has the following prototype:

```
UINT ux_host_stack_hcd_register(CHAR_PTR hcd_name,
     UINT (*hcd_initialize_function)(struct UX_HCD_STRUCT *),
           ULONG hcd_param1,
           ULONG hcd_param2);
```

The **ux_host_stack_hcd_register** function has the following parameters:

`hcd_name` String of the controller name

`hcd_initialize_function` Initialization function of the controller

`hcd_param1` Usually the IO value or memory used by the controller

`hcd_param2` Usually the IRQ used by the controller

The following are used in the definition of the OHCI controller example:

`ux_hcd_ohci` Name of the OHCI controller

`ux_ohci_initialize` Initialization routine for the OHCI controller

`0xd0000` Address at which the OHCI controller registers are visible in memory

`0x0a` IRQ used by the OHCI controller

USBX currently supports OHCI, ISP1161, and EHCI controllers. Other controllers will be added in the future.

The following is an example of the initialization of USBX in host mode with one OHCI controller and several classes:

```
UINT  status;

/* Initialize USBX. */
ux_system_initialize(memory_ptr, (128*1024),0,0);

/* The code below is required for installing the host portion of USBX.  */
status = ux_host_stack_initialize(UX_NULL);
/* If status equals UX_SUCCESS, host stack has been initialized. */

/* Register all the host classes for this USBX implementation. */
status = ux_host_class_register("ux_host_class_hub" ,
                                ux_host_class_hub_entry);
/* If status equals UX_SUCCESS, host class has been registered. */

status = ux_host_class_register("ux_host_class_storage",
                                ux_host_class_storage_entry);
/* If status equals UX_SUCCESS, host class has been registered. */

status = ux_host_class_register("ux_host_class_printer",
                                ux_host_class_printer_entry);
/* If status equals UX_SUCCESS, host class has been registered. */

status = ux_host_class_register("ux_host_class_audio",
                                ux_host_class_audio_entry);
/* If status equals UX_SUCCESS, host class has been registered. */

/* Register all the USB host controllers available in this system. */
status = ux_host_stack_hcd_register("ux_hcd_ohci",
                                ux_hcd_ohci_initialize, 0x300000, 0x0a);
/* If status equals UX_SUCCESS, USB host controllers have been registered.
```

Definition of Host Classes

One or more classes must be defined with USBX. A USB class is required to drive a USB device after the USB stack has configured the USB device. A USB class is very specific to the device. One or more classes may be required to drive a USB device,

depending on the number of interfaces contained in the USB device descriptors.

This is an example of the registration of the HUB class :

```
status = ux_host_class_register("ux_host_class_hub",ux_host_class_hub_entry
```

where

"ux_host_class_hub"
: is name of the HUB class.

ux_host_class_hub_entry
: is entry point of the HUB class.

The function **ux_host_class_register** has the following prototype:

```
UINT ux_host_class_register(CHAR_PTR class_name,
     UINT (*class_entry_address)(struct UX_HOST_CLASS_COMMAND_STRUCT *))
```

where

class_name
: is name of the class.

class_entry_address
: is entry point of the class.

Definition of USB Device Controller

Only one USB device controller can be defined at any time to operate in device mode. The application initialization file should contain this definition. The example below refers to the OKI USB device controller. For other controllers, the function entry definition has to be changed accordingly.

The following line performs the definition of an OKI controller:

```
ux_dcd_ml6965_initialize(0x7BB00000, 0, 0xB7A00000);
```

The USB device initialization has the following prototype:

```
UINT ux_dcd_ml6965_initialize (ULONG dcd_io, ULONG dcd_irq, ULONG dcd_vbus_address)
```

with the following parameters:

`ULONG dcd_io` Address of the controller IO

`ULONG dcd_irq` Interrupt used by the controller

`ULONG dcd_vbus_address` Address of the VBUS GPIO

The following example is the initialization of USBX in device mode with the storage device class and the OKI controller:

```
/* Initialize USBX Memory */
ux_system_initialize(memory_pointer,(128*1024), 0, 0);

/* The code below is required for installing the device portion of USBX */
status = _ux_device_stack_initialize(&device_framework_high_speed,
                                DEVICE_FRAMEWORK_LENGTH_HIGH_SPEED,
                                &device_framework_full_speed, DEVICE_FRAMEWORK_LENGTH_FULL_SPEED,
                                &string_framework, STRING_FRAMEWORK_LENGTH,
                                &language_id_framework, LANGUAGE_ID_FRAMEWORK_LENGTH,
                                UX_NULL);
/* If status equals UX_SUCCESS, installation was successful. */

/* Store the number of LUN in this device storage instance : single LUN.  */
   storage_parameter.ux_slave_class_storage_parameter_number_lun = 1;

/* Initialize the storage class parameters for reading/writing to the Flash Disk.  */
   storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storage_media_last_lba
        = 0x1e6bfe;
   storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storage_media_block_length
        =  512;
storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storage_media_type
        =  0;
   storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storage_media_removable_flag
        =  0x80;
   storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storage_media_read
        =  tx_demo_thread_flash_media_read;
   storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storage_media_write
        =  tx_demo_thread_flash_media_write;
   storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storage_media_status
        =  tx_demo_thread_flash_media_status;


/* Initialize the device storage class. The class is connected with interface 0 */
status =  _ux_device_stack_class_register(_ux_system_slave_class_storage_name,
                                          _ux_device_class_storage_entry,
                                          _ux_device_class_storage_thread,0,
                                           (VOID *)&storage_parameter);

/* Register the OKI USB device controllers available in this system */
status = ux_dcd_ml6965_initialize(0x7BB00000, 0, 0xB7A00000);
/* If status equals UX_SUCCESS, registration was successful. */
```

Troubleshooting

USBX is delivered with a demonstration file and a simulation environment. It is always a good idea to get the demonstration platform running first—either on the target hardware or a specific demonstration platform.

If the demonstration system does not work, try the following things to narrow the problem:

- Determine how much of the demonstration is running.
- Increase stack sizes (this is more important in actual application code than it is for the demonstration).
- Check to see if there are any error codes returned by each components installation of USBX (resources, stack/device initialization, classes initialization, controller initialization)
- Check to see if there is activity on the USB, this can be done by using a USB analyzer. If you do not have a USB analyzer, Express Logic may be able to loan you one for your testing.
- Check to see if the controller(s) interrupt handler is being called.

USBX Version ID

The current version of USBX is available both to the user and the application software during runtime. The programmer can obtain the USBX version from examination of the **usbx.txt** file. In addition, this file also contains a version history of the corresponding port. Application software can obtain the USBX version by examining the global string ***_ux_version_id***, which is defined in **ux_port.h**.

CHAPTER 3

Functional Components of USBX Host Stack

This chapter contains a description of the high-performance USBX embedded USB host stack from a functional perspective. This includes the following:

Execution Overview

The following illustrates the USBX host stack structure:

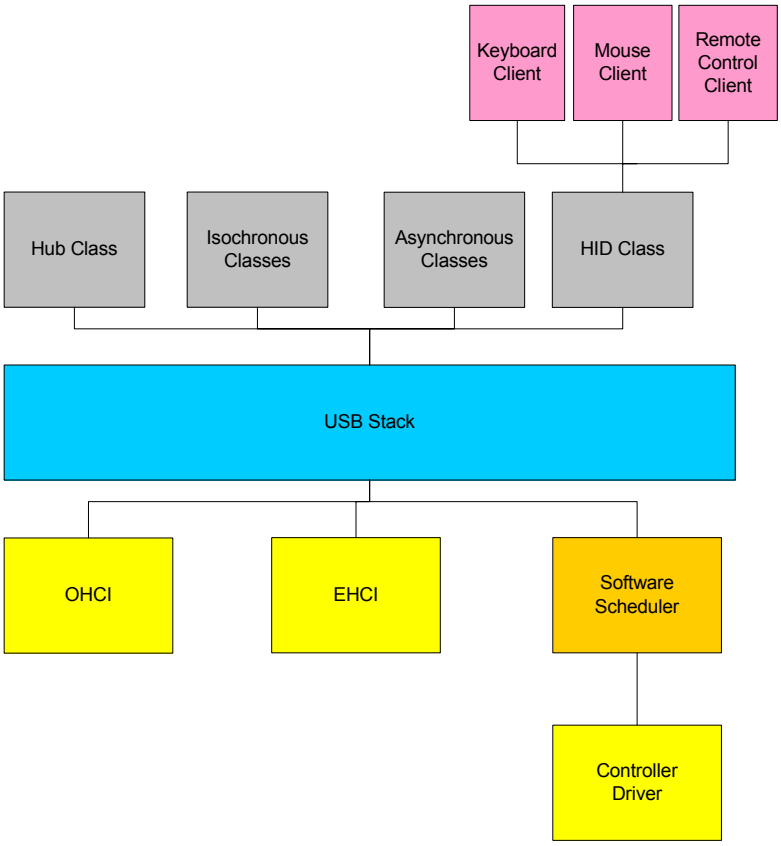


FIGURE 3. USB Host Stack

Initialization

To activate USBX, the function **ux_system_initialize** must be called. This function initializes the memory resources of USBX. To activate USBX host facilities, the function **ux_host_stack_initialize** must be called. This function will in turn initialize all the resources used by the USBX host stack such as ThreadX threads, mutexes, and semaphores.

The application initialization must activate at least one USB host controller and one or more USB classes. After the classes have been registered to the stack and the host controller(s) initialization function has been called, the bus is active and device discovery can begin. If the root hub of the host controller detects an attached device, the USB enumeration thread, which is in charge of the USB topology, will wake up and proceed to enumerate the device(s).

Because of the nature of the root hub and downstream hubs, it is possible that all attached USB devices may not have been configured completely when the host controller initialization function returns. It may take several seconds to enumerate all USB devices, especially if there are one or more hubs between the root hub and USB devices.

Application Interface Calls

There are two levels of APIs in USBX:

- USB Host Stack APIs
- USB Host Class APIs

Normally, the USBX application should not have to call any of the USB host or device stack APIs. Most applications will only access the USB Class APIs.

USB Host Stack APIs

The host stack APIs are responsible for the registration of USBX components (host classes and host

controllers), configuration of devices, and the transfer requests for available device endpoints.

USB Host Class APIs

The host class APIs are specific to each of the USB classes. Most of the common APIs for USB classes provide services such as opening/closing a device and reading from and writing to a device.

Root Hub

Each host controller instance has one or more USB root hubs. The number of root hubs is either determined by the nature of the controller or can be retrieved by reading specific registers from the controller.

Hub Class

The hub class drives the USB hubs. A USB hub can either be a standalone hub or a part of a compound device such as a keyboard or a monitor. A hub can be self powered or bus powered. Bus-powered hubs have a maximum of four downstream ports and can only allow for the connection of devices that are either self powered or bus-powered devices that use less than 100mA of power. Hubs can be cascaded. Up to five hubs can be connected to one another.

USB Host Stack

The USB stack is the center piece of USBX. It does the following:

- Manages the topology of the USB.
- Binds a USB device to one or more classes.
- Provides an API to classes to perform device descriptor interrogation and USB transfers.

Topology Manager

The USB stack topology thread is activated when a new device is connected or when a device has been disconnected. Either the root hub or a regular hub can accept device connections. After a device has been connected to the USB, the topology manager retrieves the device descriptor. This descriptor contains the number of possible configurations available for this device. Most devices have one configuration only. Some devices operate differently according to the power available on the port in which the device is connected. If this is the case, the device has multiple configurations that can be selected depending on the available power. After the device is configured by the topology manager, it is allowed to draw the amount of power specified in its configuration descriptor.

USB Class Binding

After the device is configured, the topology manager allows the class manager to continue device discovery by examining the device interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for

audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID/ vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/ Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class, such as printer, hub, storage, audio, or HID.

The class manager contains a list of registered classes from the initialization of USBX. The class manager calls each class one at a time until one class accepts management of the interface for that device. A class can only manage one interface. For example in the case of a USB audio speaker, the class manager calls all the classes for each of the interfaces.

After a class accepts an interface, a new instance of that class is created. The class manager then searches for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be used as default until a class changes it.

For the default alternate setting, the class manager mounts all endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager completes its job by returning to the class that will finish the initialization of the interface.

USBX APIs

The USB stack exports a certain number of APIs for the USB classes to interogate the device and make USB transfers on specific endpoints. These APIs are described in detail in Chapter 4.

Host Controller

The host controller driver drives a specific type of USB controller. A USB host controller can have multiple controllers inside. For instance, certain Intel PC chipsets contain two UHCI controllers. Some USB 2.0 controllers contain multiple instances of OHCI controllers in addition to one instance of the EHCI controller.

The host controller manages multiple instances of the same controller only. To drive most USB 2.0 host controllers, it must initialize both the OCHI controller and the EHCI controller during USBX initialization.

The host controller is responsible for managing the following:

- Root hub
- Power management
- Endpoints
- Transfers

Root Hub

Root hub management powers up each controller port and determines if there is a device inserted or not. This functionality is used by the USBX generic root hub to interrogate the controller downstream ports.

Power Management

Power management processing provides for the handling of suspend/resume signals either in gang mode, therefore affecting all the controller downstream ports at the same time, or individually, if the controller offers this functionality.

Endpoints

Endpoint management creates or destroys the physical endpoints to the controller. The physical endpoints are memory entities that are either parsed by the controller, if the controller supports master DMA, or written in the controller. The physical endpoints contain transaction information to be performed by the controller.

Transfers

Transfer management provides for a class to perform a transaction on each of the endpoints that have been created. Each logical endpoint contains a component called TRANSFER REQUEST for USB transfer request. The TRANSFER REQUEST is used by the stack to describe the transaction. This TRANSFER REQUEST is then passed to the stack and to the controller, which may divide it into several subtransactions depending on the capabilities of the controller.

USB Device Framework

A USB device is represented by a tree of descriptors. There are six main types of descriptors:

- Device descriptors
- Configuration descriptors
- Interface descriptors
- Endpoint descriptors

- String descriptors
- Functional descriptors

A USB device may have a simple description. Figure 4 illustrates such a device.

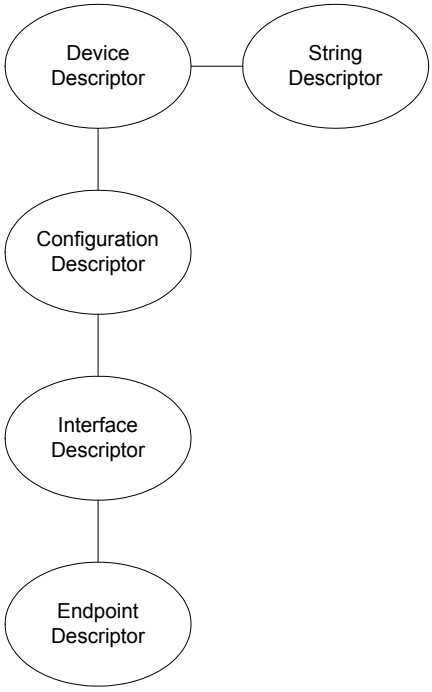


FIGURE 4. Simple USB Device

In Figure 4, the device has one configuration only. A single interface is attached to this configuration, indicating that the device has one function only and it has one endpoint only. Attached to the device descriptor is a string descriptor providing a visible identification of the device.

A device may be more complex. Figure 5 illustrates a complex USB Device.

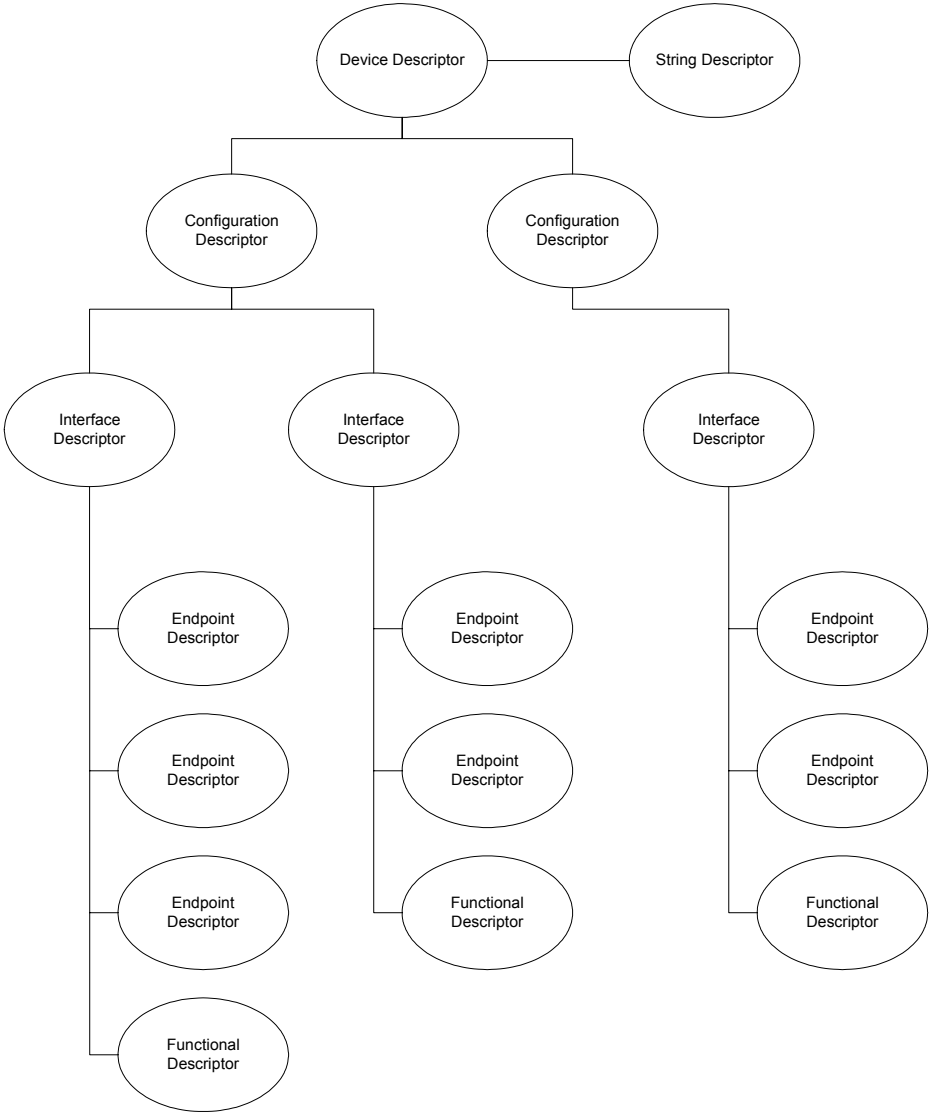


FIGURE 5. Complex USB Device

In Figure 5, the device has two configuration descriptors attached to the device descriptor. This device may indicate that it has two power modes or can be driven by either standard classes or proprietary classes. Attached to the first configuration are two interfaces indicating that the device has two logical functions. The first function has three endpoint descriptors and a functional descriptor. The functional descriptor may be used by the class responsible to drive the interface to obtain further information about this interface normally not found by a generic descriptor.

Device Descriptors

Each USB device has one single device descriptor. This descriptor contains the device identification, the number of configurations supported, and the characteristics of the default control endpoint used for configuring the device. The following table lists the device descriptor offsets, fields, size, values, and descriptions.

Offset	Field	Size	Value	Description
0	**BLength**	1	Number	Size of this descriptor in bytes
1	**bDescriptorType**	1	Constant	DEVICE Descriptor Type
2	**bcdUSB**	2	BCD	USB specification release number in binary-coded decimal For example, 2.10 is equivalent to 0x210 in BCD. This field identifies the release of the USB specification that the device and its descriptors are compliant with.

4	**bDeviceClass**	1	Class	Class code (assigned by USB-IF). If this field is reset to 0, each interface within a configuration specifies its own class information and the various interfaces operate independently. If this field is set to a value between 1 and 0xFE, the device supports different class specifications on different interfaces and the interfaces may not operate independently. This value identifies the class definition used for the aggregate interfaces. If this field is set to 0xFF, the device class is vendor specific.
5	**bDeviceSubClass**	1	SubClass	Subclass code (assigned by USB-IF). These codes are qualified by the value of the bDeviceClass field. If the bDeviceClass field is reset to 0, this field must also be reset to 0. If the bDeviceClass field is not set to 0xFF, all values are reserved for assignment by USB.

6	**bDeviceProtocol**	1	Protocol	Protocol code (assigned by USB-IF). These codes are qualified by the value of the bDeviceClass and the bDeviceSubClass fields. If a device supports class-specific protocols on a device basis as opposed to an interface basis, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to 0, the device does not use class specific protocols on a device basis. However, it may use class-specific protocols on an interface basis. If this field is set to 0xFF, the device uses a vendor-specific protocol on a device basis.
7	**bMaxPacketSize0**	1	Number	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid)
8	**idVendor**	2	ID	Vendor ID (assigned by USB-IF)
10	**idProduct**	2	ID	Product ID (assigned by the Manufacturer)
12	**bcdDevice**	2	BCD	Device release number in binary-coded decimal
14	**iManufacturer**	1	Index	Index of string descriptor describing manufacturer
15	**iProduct**	1	Index	Index of string descriptor describing product
16	**iSerialNumber**	1	Index	Index of string descriptor describing the device's serial number
17	**bNumConfigurations**	1	Number	Number of possible configurations

USBX defines a USB device descriptor as follows:

```
typedef struct UX_DEVICE_DESCRIPTOR_STRUCT
{
      UINT        bLength;
      UINT        bDescriptorType;
      USHORT      bcdUSB;
      UINT        bDeviceClass;
      UINT        bDeviceSubClass;
      UINT        bDeviceProtocol;
      UINT        bMaxPacketSize0;
      USHORT      idVendor;
      USHORT      idProduct;
      USHORT      bcdDevice;
      UINT        iManufacturer;
      UINT        iProduct;
      UINT        iSerialNumber;
      UINT        bNumConfigurations;
}  UX_DEVICE_DESCRIPTOR;
```

The USB device descriptor is part of a device container described in the following:

```
typedef struct UX_DEVICE_STRUCT
{
      ULONG                               ux_device_handle;
      ULONG                               ux_device_type;
      ULONG                               ux_device_state;
      ULONG                               ux_device_address;
      ULONG                               ux_device_speed;
      ULONG                               ux_device_port_location;
      ULONG                               ux_device_max_power;
      ULONG                               ux_device_power_source;
      ULONG                               ux_device_current_configuration;
      struct UX_DEVICE_STRUCT             *ux_device_parent;
      struct UX_HOST_CLASS_STRUCT         *ux_device_class;
      VOID                                *ux_device_class_instance;
      struct UX_HCD_STRUCT                *ux_device_hcd;
      struct UX_CONFIGURATION_STRUCT      ux_device_first_configuration;
      struct UX_DEVICE_STRUCT             *ux_device_next_device;
      struct UX_DEVICE_DESCRIPTOR_STRUCT ux_device_descriptor;
      struct UX_ENDPOINT_STRUCT           ux_device_control_endpoint;
      } UX_DEVICE;
```

The following table lists the variable names and meanings:

Variable Name	Variable Description
ux_device_handle	Device handle. Typically the address of the instance of this structure for the device
ux_device_type	Device type.
ux_device_state	Device state, which can have one of the following values: UX_DEVICE_RESET 0 UX_DEVICE_ATTACHED 1 UX_DEVICE_ADDRESSED 2 UX_DEVICE_CONFIGURED 3 UX_DEVICE_SUSPENDED 4
ux_device_address	Address of the device after the SET_ADDRESS command has been accepted (from 1 to 127)
ux_device_speed	Speed of the device: UX_LOW_SPEED_DEVICE 0 UX_FULL_SPEED_DEVICE 1 UX_HIGH_SPEED_DEVICE 2
ux_device_port_location	Index of the port of the parent device (root hub or hub)
ux_device_max_power	Maximum power in mA that the device may take in the selected configuration
ux_device_power_source	Can be one of the following values: UX_DEVICE_BUS_POWERED 1 UX_DEVICE_SELF_POWERED 2
ux_device_current_configuration	Index of current configuration being used by device
ux_device_parent	Device container pointer of parent of this device. If the pointer is null, the parent is root hub of the controller.
ux_device_class	Pointer to the class type that owns this device
ux_device_class_instance	Pointer to the instance of the class that owns this device
ux_device_hcd	USB host controller instance to which this device is attached

ux_device_first_configuration	Pointer to the first configuration container for this device
ux_device_next_device	Pointer to next device in the list of devices on any of the buses detected by USBX
ux_device_descriptor	USB device descriptor
ux_device_control_endpoint	Descriptor of the default control endpoint used by this device.

Configuration Descriptors

The configuration descriptor describes information about a specific device configuration. A USB device may contain one or more configuration descriptors. The ***bNumConfigurations*** field in the device descriptor indicates the number of configuration descriptors.The descriptor contains a ***bConfigurationValue*** field with a value that, when used as a parameter to the Set Configuration request, causes the device to assume the described configuration.

The descriptor describes the number of interfaces provided by the configuration. Each interface represents a logical function within the device and may operate independently. For instance, a USB audio speaker may have three interfaces: one for audio streaming, one for audio control, and one HID interface to manage the speaker's buttons.

When the host issues a **GET_DESCRIPTOR** request for the configuration descriptor, all related interface and endpoint descriptors are returned.

The following table lists the offsets, fields, sizes, values, and descriptions of the configuration descriptors:

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor in bytes
1	**bDescriptorType**	1	Constant	CONFIGURATION
2	**wTotalLength**	2	Number	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class or vendor specific)
4	**bNumInterfaces**	1	Number	Number of interfaces supported by this configuration
5	**bConfigurationValue**	1	Number	Value to use as an argument to Set Configuration
6	**iConfiguration**	1	Index	Index of string descriptor describing this configuration

7	**bMAttributes**	1	Bitmap	Configuration characteristics D7 Bus Powered D6 Self Powered D5 Remote Wakeup D4..0 Reserved (reset to 0) A device configuration that uses power from the bus and a local source sets both D7 and D6. The actual power source at runtime may be determined using the Get Status device request. If a device configuration supports remote wakeup, D5 is set to 1.
8	**MaxPower**	1	mA	Maximum power consumption of USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2 mA units (i.e., 50 = 100 mA). Note: A device configuration reports if the configuration is bus-powered or self-powered. Device status reports if the device is currently self-powered. If a device is disconnected from its external power source, it updates device status to indicate it is no longer self-powered.

USBX defines a USB configuration descriptor as follows:

```
typedef struct UX_CONFIGURATION_DESCRIPTOR_STRUCT
{
      UINT        bLength;
      UINT        bDescriptorType;
      USHORT      wTotalLength;
      UINT        bNumInterfaces;
      UINT        bConfigurationValue;
      UINT        iConfiguration;
      UINT        bmAttributes;
      UINT        MaxPower;
      } UX_CONFIGURATION_DESCRIPTOR;
```

The USB configuration descriptor is part of a configuration container described as follows:

```
typedef struct UX_CONFIGURATION_STRUCT
{
      ULONG                                   ux_configuration_handle;
      ULONG                                   ux_configuration_state;
      struct UX_CONFIGURATION_DESCRIPTOR_STRUCTux_configuration_descriptor;
      struct UX_INTERFACE_STRUCT              *ux_configuration_first_interface;
      struct UX_CONFIGURATION_STRUCT          *ux_configuration_next_configuration;
      struct UX_DEVICE_STRUCT                 *ux_configuration_device;
UX_CONFIGURATION;
```

The following table lists the configuration descriptor variable names and descriptions:

Variable Name	Variable Description
ux_configuration_handle	Handle of the configuration. This is typically the address of the instance of this structure for the configuration
ux_configuration_state	State of the configuration
ux_configuration_descriptor	USB device descriptor
ux_configuration_first_interface	Pointer to the first interface for this configuration

ux_configuration_next_configuration	Pointer to the next configuration for the same device
ux_configuration_device	Pointer to the device owner of this configuration

Interface Descriptors

The interface descriptor describes a specific interface within a configuration. An interface is a logical function within a USB device. A configuration provides one or more interfaces, each with zero or more endpoint descriptors describing a unique set of endpoints within the configuration. When a configuration supports more than one interface, the endpoint descriptors for a particular interface follow the interface descriptor in the data returned by the **GET_DESCRIPTOR** request for the specified configuration.

An interface descriptor is always returned as part of a configuration descriptor. An interface descriptor cannot be directly access by a **GET_DESCRIPTOR** request.

An interface may include alternate settings that allow the endpoints and their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. A class can select to change the current alternate setting to change the interface behavior and the characteristics of the associated endpoints. The **SET_INTERFACE** request is used to select an alternate setting or to return to the default setting.

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

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

An interface may not have any endpoints associated with it, in which case, only the default control endpoint is valid for that interface.

Alternate settings are used mainly to change the requested bandwidth for periodic endpoints associated with the interface. For example, a USB speaker streaming interface should have the first alternate setting with a 0 bandwidth demand on its isochronous endpoint. Other alternate settings may select different bandwidth requirements, depending on the audio streaming frequency.

The USB descriptor for the interface is as follows:

Offset	Field	Size	Value	Descriptor
0	***bLength***	1	Number	Size of this descriptor in bytes
1	***bDescriptorType***	1	Constant	INTERFACE descriptor type
2	***bInterfaceNumber***	1	Number	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration
3	***bAltenateSetting***	1	Number	Value used to select alternate setting for the interface identified in the prior field

4	***bNumEndpoints***	1	Number	Number of endpoints used by this interface (excluding endpoint zero). If this value is 0, this interface only uses endpoint zero.
5	***bInterfaceClass***	1	Class	Class code (assigned by USB). If this field is reset to 0, the interface does not belong to any USB specified device class. If this field is set to 0xFF, the interface class is vendor specific. All other values are reserved for assignment by USB.
6	***bInterfaceSubClass***	1	SubClass	Subclass code (assigned by USB). These codes are qualified by the value of the bInterfaceClass field. If the bInterfaceClass field is reset to 0, this field must also be reset to 0. If the bInterfaceClass field is not set to 0xFF, all values are reserved for assignment by USB.
7	***bInterfaceProtocol***	1	Protocol	Protocol code (assigned by USB). These codes are qualified by the value of the bInterfaceClass and the bInterfaceSubClass fields. If an interface supports class-specific requests, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to 0, the device does not use a class specific protocol on this interface. If this field is set to 0xFF, the device uses a vendor specific protocol for this interface.
8	***iInterface***	1	Index	Index of string descriptor describing this interface

USBX defines a USB interface descriptor as follows:

```
typedef struct UX_INTERFACE_DESCRIPTOR_STRUCT
{
        UINT        bLength;
        UINT        bDescriptorType;
        UINT        bInterfaceNumber;
        UINT        bAlternateSetting;
        UINT        bNumEndpoints;
        UINT        bInterfaceClass;
        UINT        bInterfaceSubClass;
        UINT        bInterfaceProtocol;
        UINT        iInterface;
} UX_INTERFACE_DESCRIPTOR;
```

The USB interface descriptor is part of an interface container described as follows:

```
typedef struct UX_INTERFACE_STRUCT
{
       ULONG                          ux_interface_handle;
       ULONG                          ux_interface_state;
       ULONG                          ux_interface_current_alternate_setti
       struct UX_INTERFACE_DESCRIPTOR_STRUCT
                                      ux_interface_descriptor;
       struct UX_HOST_CLASS_STRUCT    *ux_interface_class;
       VOID                           *ux_interface_class_instance;
       struct UX_ENDPOINT_STRUCT      *ux_interface_first_endpoint;
       struct UX_INTERFACE_STRUCT     *ux_interface_next_interface;
       struct UX_CONFIGURATION_STRUCT *ux_interface_configuration;
} UX_INTERFACE;
```

The following table lists the interface descriptor variable names and descriptions:

Variable Name	Variable Description
ux_interface_handle	Handle of the interface. Typically the address of the instance of this structure for the interface
ux_interface_state	State of the interface
ux_interface_descriptor	USB interface descriptor
ux_interface_class	Pointer to the class type that owns this interface

ux_interface_class_instance	Pointer to the instance of the class that owns this interface
ux_interface_first_endpoint	Pointer to the first endpoint registered with this interface
ux_interface_next_interface	Pointer to the next interface associated with the configuration
ux_interface_configuration	Pointer to the configuration owner of this interface

Endpoint Descriptors

Each endpoint associated with an interface has its own endpoint descriptor. This descriptor contains the information required by the host stack to determine the bandwidth requirements of each endpoint, the maximum payload associated with the endpoint, its periodicity, and its direction. An endpoint descriptor is always returned by a GET_DESCRIPTOR command for the configuration.

The default control endpoint associated to the device descriptor is not counted as part of the endpoint(s) associated with the interface and, therefore, not returned in this descriptor.

When the host software requests a change of the alternate setting for an interface, all the associated endpoints and their USB resources are modified according to the new alternate setting.

Except for the default control endpoints, endpoints cannot be shared between interfaces.

The following table lists the endpoint descriptor offsets, fields, size, values, and descriptions:

Offset	Field	Size	Value	Description
0	***bLength***	1	Number	Size of this descriptor in bytes
1	***bDescriptorType***	1	Constant	ENDPOINT descriptor type

2	***bEndpointAddress***	1	Endpoint	Address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 3...0: The endpoint number Bit 6...4: Reserved, reset to zero Bit 7: Direction, ignored for control endpoints 0 = OUT endpoint 1 = IN endpoint
3	***bmAttributes***	1	Bitmap	Describes the endpoint's attributes when it is configured using the bConfigurationValue. Bits 1..0: Transfer Type 00 = Control 01 = Isochronous 10 = Bulk 11 = Interrupt If not an isochronous endpoint, bits 5..2 are reserved and must be set to zero. If isochronous, they are defined as follows: Bits 3..2: Synchronization Type 00 = No Synchronization 01 = Asynchronous 10 = Adaptive 11 = Synchronous Bits 5..4: Usage Type 00 = Data endpoint 01 = Feedback endpoint 10 = Implicit feedback Data endpoint 11 = Reserved

4	***wMaxPacketSize***	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. For isochronous endpoints, this value is used to reserve the bus time in the schedule, required for the per-(micro)frame data payloads. The pipe may, on an ongoing basis, actually use less bandwidth than that reserved. The device reports, if necessary, the actual bandwidth used via its normal, non-USB defined mechanisms. For all endpoints, bits 10..0 specify the maximum packet size (in bytes). For high-speed isochronous and interrupt endpoints: Bits 12..11 specify the number of additional transaction opportunities per microframe: 00 = None (1 transaction per microframe) 01 = 1 additional (2 per microframe) 10 = 2 additional (3 per microframe) 11 = Reserved Bits 15..13 are reserved and must be set to zero.

6	***bInterval***	1	Number	Number interval for polling endpoint for data transfers. Expressed in frames or microframes depending on the device operating speed (i.e., either 1 millisecond or 125 µs units). For full-/high-speed isochronous endpoints, this value must be in the range from 1 to 16. The *bInterval* value is used as the exponent for a 2bInterval-1 value; e.g., a *bInterval* of 4 means a period of 8 (24-1). For full-/low-speed interrupt endpoints, the value of this field may be from 1 to 255. For high-speed interrupt endpoints, the *bInterval* value is used as the exponent for a 2bInterval-1 value; e.g., a *bInterval* of 4 means a period of 8 (24-1). This value must be from 1 to 16. For high-speed bulk/control OUT endpoints, the *bInterval* must specify the maximum NAK rate of the endpoint. A value of 0 indicates the endpoint never NAKs. Other values indicate at most 1 NAK each *bInterval* number of microframes. This value must be in the range from 0 to 255.

USBX defines a USB endpoint descriptor as follows:

```
typedef struct UX_ENDPOINT_DESCRIPTOR_STRUCT
{
      UINT        bLength;
      UINT        bDescriptorType;
      UINT        bEndpointAddress;
      UINT        bmAttributes;
      USHORT      wMaxPacketSize;
      UINT        bInterval;
} UX_ENDPOINT_DESCRIPTOR;
```

The USB endpoint descriptor is part of an endpoint container described as follows:

```
typedef struct UX_ENDPOINT_STRUCT
{
      ULONG                                 ux_endpoint_handle;
      ULONG                                 ux_endpoint_state;
      VOID                                  *ux_endpoint_ed;
      struct UX_ENDPOINT_DESCRIPTOR_STRUCTux_endpoint_descriptor;
      struct UX_ENDPOINT_STRUCT             *ux_endpoint_next_endpoint;
      struct UX_INTERFACE_STRUCT            *ux_endpoint_interface;
      struct UX_DEVICE_STRUCT               *ux_endpoint_device;
      struct UX_TRANSFER REQUEST_STRUCT     ux_endpoint_transfer request;
} UX_ENDPOINT;
```

The following table lists the endpoint descriptor variable names and descriptions:

Variable Name	Variable Description
ux_endpoint_handle	Handle of the endpoint. This is typically the address of the instance of this structure for the endpoint
ux_endpoint_state	State of the endpoint
ux_endpoint_ed	Pointer to the physical endpoint at the host controller layer
ux_endpoint_descriptor	USB endpoint descriptor
ux_endpoint_next_endpoint	Pointer to the next endpoint that belongs to the same interface
ux_endpoint_interface	Pointer to the interface that owns this endpoint interface

ux_endpoint_device	Pointer to the parent device container
ux_endpoint_transfer request	USB transfer request used to send/receive data from to/from the device

String Descriptors

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

String descriptors use UNICODE encodings that allows support for several character sets. The strings in a USB device may support multiple languages. When requesting a string descriptor, the requester specifies the desired language using a language ID defined by the USB-IF. The list of currently defined USB LANGIDs can be found in Appendix D. String index zero for all languages returns a string descriptor that contains an array of two-byte LANGID codes supported by the device. Note that the UNICODE string is not 0 terminated. Instead, the size of the string array is computed by subtracting two from the size of the array contained in the first byte of the descriptor.

The USB string descriptor 0 is encoded as follows:

Offset	Field	Size	Value	Description
0	***bLength***	1	N+2	Size of this descriptor in bytes
1	***bDescriptorType***	1	Constant	STRING descriptor type
2	***wLANGID[0]***	2	Number	LANGID code 0
..	***…]***	..	…	…
N	***wLANGID[n]***	2	Number	LANGID code n

Other USB string descriptors are encoded as follows:

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

USBX defines a non-zero USB string descriptor as follows:

```
typedef struct UX_STRING_DESCRIPTOR_STRUCT
{
      UINT        bLength;
      UINT        bDescriptorType;
      USHORT      bString[1];
} UX_STRING_DESCRIPTOR;
```

Functional Descriptors

Functional descriptors are also known as class-specific descriptors. They normally use the same basic structures as generic descriptors and allow for additional information to be available to the class. For example, in the case of the USB audio speaker, class specific descriptors allow the audio class to retrieve for each alternate setting the type of audio frequency supported.

USBX Device Descriptor Framework in Memory

USBX maintains most of a device descriptors in memory, that is all descriptors except the string and

functional descriptors. Figure 6 shows how these descriptors are stored and related:

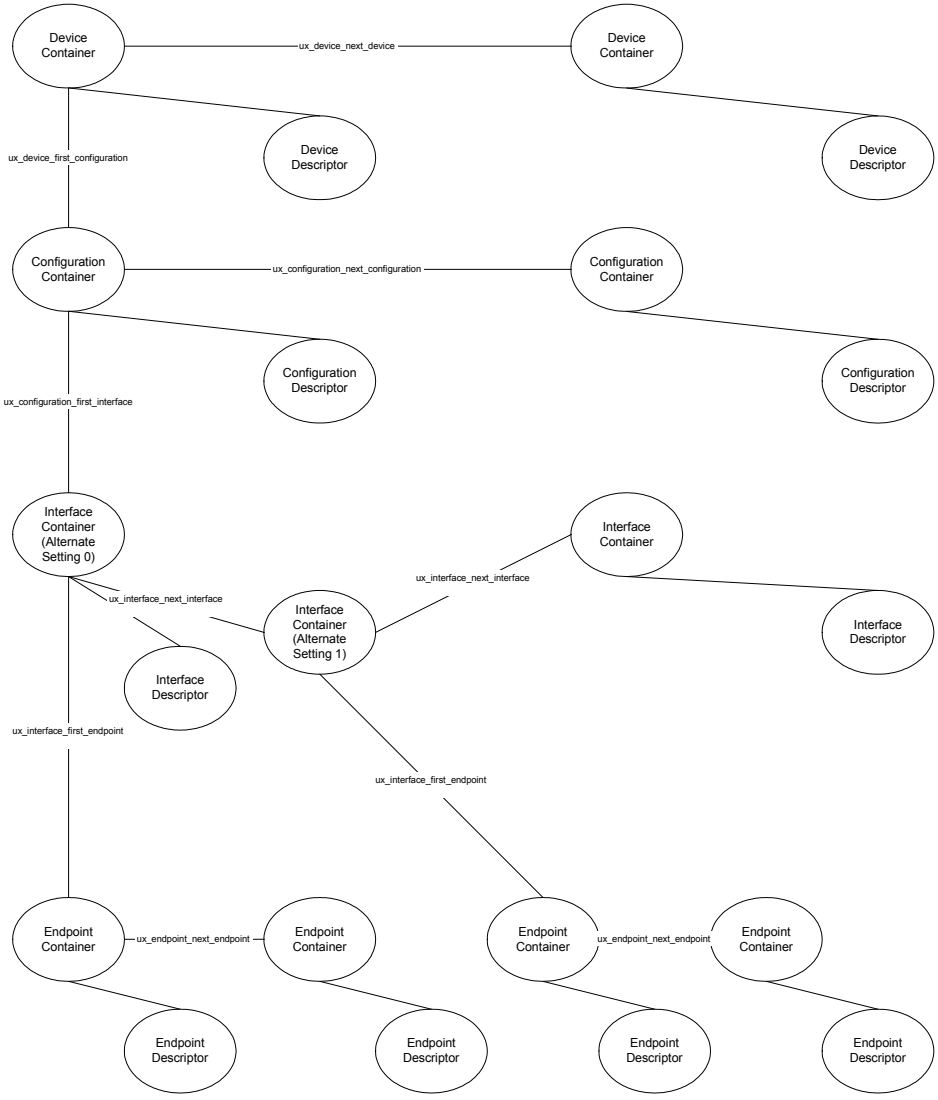


FIGURE 6. Descriptor Relationships

U S B X

CHAPTER 4

Description of USBX Host Services

This chapter contains a description of all USBX services in alphabetic order. Service names are designed so all similar services are grouped together.

U S B X

ux_host_stack_initialize

Initialize USBX host operation

Prototype

```
UINT  ux_host_stack_initialize(UINT (*system_change_function)(ULONG, UX_HOST_CLASS *))
```

Description

This function will initialize the USB host stack. The supplied memory area will be setup for USBX internal use. If UX_SUCCESS is returned, USBX is ready for host controller and class registration.

Input Parameters

system_change_function Pointer to optional callback routine for notifying application of device changes.

Return Values

UX_SUCCESS	(0x00)	Successful initialization.

Example

```
UINT   status;
/* Initialize USBX for host operation, without notification.  */
status = ux_host_stack_initialize(UX_NULL);
/* If status equals UX_SUCCESS, USBX has been successfully
      initialized for host operation. */
```

ux_host_stack_endpoint_transfer_abort

Abort transactions attached to transfer request for endpoint

Prototype

```
UINT ux_host_stack_endpoint_transfer_abort(UX_ENDPOINT *endpoint)
```

Description

This function will cancel all transactions active or pending for a specific transfer request attached to an endpoint. If the transfer request has a callback function attached, the callback function will be called with the UX_TRANSACTION_ABORTED status.

Input Parameters

endpoint	Pointer to an endpoint.

Return Values

UX_SUCCESS	(0x00)	No errors.
UX_ENDPOINT_HANDLE_UNKNOWN	(0x53)	Endpoint handle not valid.

Example

```
UX_HOST_CLASS_PRINTER   *printer;
UINT   status;

/* Get the instance for this class. */
printer = (UX_HOST_CLASS_PRINTER *) command ->
   ux_host_class_command_instance;

/* The printer is being shut down. */
printer -> printer_state = UX_HOST_CLASS_INSTANCE_SHUTDOWN;

/* We need to abort transactions on the bulk out pipe. */
status = ux_host_stack_endpoint_transfer_abort
           (printer -> printer_bulk_out_endpoint);

/* If status equals UX_SUCCESS, the operation was successful */
```

ux_host_stack_class_get

Get pointer to a class container

Prototype

```
UINT ux_host_stack_class_get(UCHAR *class_name,UX_CLASS **class)
```

Description

This function returns a pointer to the class container. A class needs to obtain its container from the USB stack to search for instances when a class or an application wants to open a device.

Input Parameters

class_name	Pointer to the class name.
class	Pointer updated by the function call that contains the class container for the name of the class.

Return Values

UX_SUCCESS	(0x00)	No errors, on return the class field is filed with the pointer to the class container.
UX_CLASS_UNKNOWN	(0x59)	Class is unknown by the stack.

Example

```
UX_HOST_CLASS   *printer_container;
UINT            status;

/* Get the container for this class.  */
status = ux_host_stack_class_get("ux_host_class_printer",
   &printer_container);
/* If status equals UX_SUCCESS, the operation was successful */
```

ux_host_stack_class_register

Register USB class to USB stack

Prototype

```
UINT ux_host_stack_class_register(CHAR_PTR class_name,
       UINT (*class_entry_address)(struct UX_CLASS_COMMAND_STRUCT *))
```

Description

This function registers a USB class to the USB stack. The class must specify an entry point for the USB stack to send commands such as :

UX_CLASS_COMMAND_QUERY

UX_CLASS_COMMAND_ACTIVATE

UX_CLASS_COMMAND_DESTROY

Input Parameters

class_name Pointer to the name of the class; valid entries include:

"ux_host_class_audio"
"ux_host_class_hid"
"ux_host_class_hid_client_remote_ control"
"ux_host_class_hid_client_mouse"
"ux_host_class_hub"
"ux_host_class_printer"
"ux_host_class_storage"

class_entry_address Address of the class function; valid entry functions include:

ux_host_class_audio_entry
ux_host_class_hid_entry
ux_host_class_hid_remote_control_entry
ux_host_class_hid_mouse_entry
ux_host_class_hub_entry
ux_host_class_printer_entry
ux_host_class_storage_entry

Return Values

UX_SUCCESS	(0x00)	Class installed successfully.
UX_MEMORY_INSUFFICIENT	(0x12)	No more memory in USBX to store this class.
UX_CLASS_ALREADY_INSTALLED	(0x58)	Host class already installed.

Example

```
UINT   status;
/* Register all the classes for this implementation. */

status = ux_host_stack_class_register("ux_host_class_hub",
                                      ux_host_class_hub_entry);
/* If status equals UX_SUCCESS, the class was successfully
   installed. */
```

ux_host_stack_class_instance_create

Create new class instance for class container

Prototype

```
UINT  ux_host_stack_class_instance_create(UX_HOST_CLASS *class, VOID *class_instance)
```

Description

This function creates a new class instance for a class container. The instance of a class is not contained in the class code to reduce the class complexity. Rather, each class instance is attached to class container located in the main stack.

Input Parameters

class	Pointer to the class container.
class_instance	Pointer to the class instance to be created.

Return Values

UX_SUCCESS	(0x00)	Class instanced attached to the class container.

Example

```
UINT                    status;
UX_HOST_CLASS_PRINTER   *printer;

/* Obtain memory for this class instance */
printer=ux_memory_allocate(UX_NO_ALIGN,sizeof(UX_HOST_CLASS_PRINTER
   ));
if(printer==UX_NULL)return(UX_MEMORY_INSUFFICIENT);

/* Store the class container into this instance */
printer->printer_class=command->ux_host_class;

/* Create this class instance */
status=ux_host_stack_class_instance_create(printer->printer_class,
                                           (VOID *)printer);

/* If status equals UX_SUCCESS, the class instance was successfully
   and attached to the class container. /*
```

ux_host_stack_class_instance_destroy

Destroy class instance for class container

Prototype

```
UINT ux_host_stack_class_instance_destroy(UX_HOST_CLASS *class,VOID *class_instance)
```

Description

This function destroys a class instance for a class container.

Input Parameters

class	Pointer to class container.
class_instance	Pointer to instance to destroy.

Return Values

UX_SUCCESS	(0x00)	Class instance destroyed.
UX_CLASS_INSTANCE_UNKNOWN	(0x5b)	Class instance not attached to class container.

Example

```
UINT                    status;
UX_HOST_CLASS_PRINTER   *printer;

/* Get the instance for this class.  */
printer = (UX_HOST_CLASS_PRINTER *) command ->
  ux_host_class_command_instance;

/* The printer is being shut down.  */
printer -> printer_state = UX_HOST_CLASS_INSTANCE_SHUTDOWN;

/* Destroy the instance. */
status = ux_host_stack_class_instance_destroy(printer -> printer_class,
                                              (VOID *)printer);

/* If status equals UX_SUCCESS, the class instance was successfully
  destroyed. */
```

ux_host_stack_class_instance_get

Get class instance pointer for specific class

Prototype

```
UINT ux_host_stack_class_instance_get(UX_HOST_CLASS *class,UINT
                                      class_index, VOID **class_instance)
```

Description

This function returns a class instance pointer for a specific class. The instance of a class is not contained in the class code to reduce the class complexity. Rather, each class instance is attached to the class container. This function is used to search for class instances within a class container.

Input Parameters

class	Pointer to class container.
class_index	Index to be used by the function call within list of attached classes to the container.
class_instance	Pointer to instance to be returned by function call.

Return Values

UX_SUCCESS	(0x00)	Class instance found.
UX_HOST_CLASS_INSTANCE_UNKNOWN	(0x5b)	No more classes instances attached to the class container.

Example

```
UINT                        status
UX_HOST_CLASS_PRINTER       *printer;

/* Obtain memory for this class instance */
printer=ux_memory_allocate(UX_NO_ALIGN,sizeof(UX_HOST_CLASS_PRINTER));
if(printer==UX_NULL)
    return(UX_MEMORY_INSUFFICIENT);

/* Search for instance index 2 */
status=ux_host_stack_class_instance_get(class,2,(VOID *)printer);

/* If status equals UX_SUCCESS, the class instance was found. */
```

ux_host_stack_device_configuration_get

Get a pointer to a configuration container

Prototype

```
UINT ux_host_stack_configuration_get(UX_DEVICE *device,
                                     UINT configuration_index,
                                     UX_CONFIGURATION **configuration)
```

Description

This function returns a configuration container based on a device handle and a configuration index.

Input Parameters

device	Pointer to device container that owns configuration requested.
configuration_index	Index of configuration to be searched.
configuration	Address of pointer to configuration container to be returned.

Return Values

UX_SUCCESS	(0x00)	Configuration found.
UX_DEVICE_HANDLE_UNKNOWN	(0x50)	Device container does not exist.
UX_CONFIGURATION_HANDLE_UNKNOWN	(0x51)	Configuration handle for index does not exist.

Example

```
UINT                            status;
UX_HOST_CLASS_PRINTER           *printer;

/* If the device has been configured already, we don't need to do it again.
*/
if (printer -> printer_device -> ux_device_state == UX_DEVICE_CONFIGURED)
return(UX_SUCCESS);

/* A printer normally has one configuration, retrieve 1st configuration only.
*/
status = ux_host_stack_device_configuration_get(printer -> printer_device,
                                                0, configuration);

/* If status equals UX_SUCCESS, the configuration was found. */
```

ux_host_stack_device_configuration_select

Select specific configuration for a device

Prototype

```
UINT ux_host_stack_configuration_select(UX_CONFIGURATION *configuration)
```

Description

This function selects a specific configuration for a device. When this configuration is set to the device, by default, each device interface and its associated alternate setting 0 are activated on the device. If the device/ interface class needs to change the setting of a particular interface, it issues a **ux_host_stack_interface_setting_select** service call.

Input Parameters

configuration	Pointer to configuration container that is to be enabled for this device

Return Values

UX_SUCCESS	(0x00)	Configuration selection successful.
UX_CONFIGURATION_HANDLE_UNKNOWN	(0x51)	Configuration handle does not exist.
UX_OVER_CURRENT_CONDITION	(0x43)	Over current condition exists on the bus for this configuration.

Example

```
UINT                    status;
UX_HOST_CLASS_PRINTER   *printer;

/* If the device has been configured already, we don't need to do it again. */
if (printer -> printer_device -> ux_device_state == UX_DEVICE_CONFIGURED)
      return(UX_SUCCESS);

/* A printer normally has one configuration - retrieve 1st configuration only. */
status = ux_host_stack_device_configuration_get(printer -> printer_device,
                                                0,configuration);
/* If status equals UX_SUCCESS, the configuration selection was successful. */

/* If valid configuration, ask the USB stack to set this configuration. */
status = ux_host_stack_device_configuration_select(configuration);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_stack_device_get

Get pointer to a device container

Prototype

```
UINT ux_host_stack_device_get(ULONG device_index,UX_DEVICE **device)
```

Description

This function returns a device container based on its index. The device index starts with 0.

The index is a ULONG because there can be several controllers and a byte index might not be enough. The device index should not be confused with the device address, which is bus specific.

Input Parameters

device_index	Index of the device.
device	Address of pointer for device container to return.

Return Values

UX_SUCCESS	(0x00)	Device container exists and is returned.
UX_DEVICE_HANDLE_UNKNOWN	(0x50)	Device unknown.

Example

```
UINT   status;

/* Locate the first device in USBX. */

status = ux_host_stack_device_get(0, device);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_stack_interface_endpoint_get

Get endpoint container

Prototype

```
UINT ux_host_stack_interface_endpoint_get(UX_INTERFACE *interface,
                                          UINT endpoint_index,
                                          UX_ENDPOINT **endpoint)
```

Description

This function returns an endpoint container based on the interface handle and an endpoint index. It is assumed that the alternate setting for the interface has been selected or the default setting is being used prior to the endpoint(s) being searched.

Input Parameters

interface	Pointer to the interface container that contains endpoint requested.
endpoint_index	Index of endpoint in this interface.
endpoint	Address of endpoint container to be returned.

Return Values

UX_SUCCESS	(0x00)	Endpoint container exists and is returned.
UX_INTERFACE_HANDLE_UNKNOWN	(0x52)	Interface specified does not exist.
UX_ENDPOINT_HANDLE_UNKNOWN	(0x53)	Endpoint does not exist.

Example

```
UINT                    status;
UX_HOST_CLASS_PRINTER   *printer;

for(endpoint_index = 0;
    endpoint_index < printer -> printer_interface ->
                            ux_interface_descriptor.bNumEndpoints;
    endpoint_index++)
{
    status = ux_host_stack_interface_endpoint_get(printer ->printer_interface,
                endpoint_index, &endpoint);
    if (status == UX_SUCCESS)
   {
   /* Check if endpoint is bulk and OUT. */
   if (((endpoint -> ux_endpoint_descriptor.bEndpointAddress &
                 UX_ENDPOINT_DIRECTION) == UX_ENDPOINT_OUT) &&
         ((endpoint -> ux_endpoint_descriptor.bmAttributes &
               UX_MASK_ENDPOINT_TYPE) == UX_BULK_ENDPOINT))
         return(UX_SUCCESS)
    }
}
```

ux_host_stack_hcd_register

Register USB controller to USB stack

Prototype

```
UINT ux_host_stack_hcd_register(CHAR_PTR hcd_name,
                                UINT (*hcd_function)(struct UX_HCD_STRUCT *),
                                ULONG hcd_param1,
                                ULONG hcd_param2)
```

Description

This function registers a USB controller to the USB stack. It mainly allocates the memory used by this controller and passes the initialization command to the controller.

Input Parameters

hcd_name	Name of host controller; available controllers include: "ux_hcd_ohci" "ux_hcd_ehci" "ux_hcd_isp1161"
hcd_function	Function in host controller responsible for initialization. Available controller entry functions include: ux_hcd_ehci_initialize ux_hcd_ohci_initialize ux_hcd_isp1161_initialize
hcd_param1	IO or memory resource used by hcd
hcd_param2	IRQ used by host controller

Return Values

UX_SUCCESS	(0x00)	Controller initialized.
UX_MEMORY_INSUFFICIENT	(0x12)	Not enough memory for this controller.
UX_PORT_RESET_FAILED	(0x31)	Reset of controller failed.
UX_CONTROLLER_INIT_FAILED	(0x32)	Controller failed to initialize.

Example

```
UINT   status;

/* Initialize an OHCI controller mapped at address 0xd0000 and using IRQ 10. */
status = ux_host_stack_hcd_register("ux_hcd_ohci", ux_hcd_ohci_initialize,
                                                   0xd0000, 0x0a);

/* If status equals UX_SUCCESS, the controller was initialized properly. */

/* Note that the application must also setup a call to the interrupt
   handler for the OHCI controller. The function for OHCI is called
   _ux_hch_ohci_interrupt_handler. */
```

ux_host_stack_configuration_interface_get

Get interface container pointer

Prototype

```
UINT ux_host_stack_configuration_interface_get UX_CONFIGURATION *configuration,
                                               UINT interface_index,
                                               UINT alternate_setting_index,
                                               UX_INTERFACE **interface)
```

Description

This function returns an interface container based on a configuration handle, an interface index, and an alternate setting index.

Input Parameters

configuration	Pointer to configuration container that owns the interface.
interface_index	Interface index to be searched.
alternate_setting_index	Alternate setting within the interface to search.
interface	Address of interface container pointer to be returned.

Return Values

UX_SUCCESS	(0x00)	Interface container index and alternate setting found and returned.
UX_CONFIGURATION_HANDLE_UNKNOWN	(0x51)	Configuration does not exist.
UX_INTERFACE_HANDLE_UNKNOWN	(0x52)	Interface does not exist.

Example

```
UINT   status;

/* Search for the default alternate setting on the first interface for the
   printer. */

status = ux_host_stack_configuration_interface_get(configuration, 0, 0,
                                   &printer -> printer_interface);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_stack_interface_setting_select

Selects alternative setting for interface

Prototype

```
UINT ux_host_stack_configuration_interface_setting_select (UX_INTERFACE *interfa
```

This function selects a specific alternate setting for a given interface belonging to the selected configuration. It is used to change from the default alternate setting to a new setting or to go back to the default alternate setting. When a new alternate setting is selected, the previous endpoint characteristics are invalid and should be reloaded.

Input Parameters

interface	Pointer to interface container whose alternate setting is to be selected.

Return Values

UX_SUCCESS	(0x00)	Alternate setting for interface selected.
UX_INTERFACE_HANDLE_UNKNOWN	(0x52)	Interface does not exist.

Example

```
UINT   status;

/* Select a new alternate setting for this interface. */
status = ux_host_stack_interface_setting_select(interface);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_stack_transfer_request_abort

Abort pending transfer request

Prototype

```
UINT  ux_host_stack_transfer_request_abort(UX_TRANSFER REQUEST *transfer
                    request)
```

Description

This function aborts a previously submitted pending transfer request. This function only cancels a specific transfer request. The call back to the function will have the UX_TRANSFER REQUEST_STATUS_ABORT status.

Input Parameters

transfer request	Pointer to the transfer request to be aborted.

Return Values

UX_SUCCESS	(0x00)	USB transfer for this request was canceled.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_host_stack_transfer_request_abort(transfer request);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_stack_transfer_request

Request USB transfer

Prototype

```
UINT  ux_host_stack_transfer_request(UX_TRANSFER REQUEST *transfer request)
```

Description

This function performs a USB transaction. On entry the transfer request gives the endpoint pipe selected for this transaction and the parameters associated with the transfer (data payload, length of transaction). For control pipe, the transaction is blocking and will only return when the three phases of the control transfer have been completed or if there is a previous error. For other pipes, the USB stack will schedule the transaction on the USB but will not wait for its completion. Each transfer request for non-blocking pipes has to specify a completion routine handler.

When the function call returns, the status of the transfer request should be examined as it contains the result of the transaction.

Input Parameters

transfer_request	Pointer to the transfer request. The transfer request contains all the necessary information required for the transfer.

Return Values

UX_SUCCESS	(0x00)	USB transfer for this transfer request was scheduled properly. Status code of transfer request should be examined when the transfer request completes.
UX_MEMORY_INSUFFICIENT	(0x12)	Not enough memory to allocate the necessary controller resources.

Example

```
UINT   status;

/* Create a transfer request for the SET_CONFIGURATION request.
   No data for this request. */
transfer_request -> ux_transfer_endpoint_handle = control_endpoint;
transfer_request -> ux_transfer_requested_length = 0;
transfer_request -> ux_transfer_request_function = UX_SET_CONFIGURATION;
transfer_request -> ux_transfer_request_type = UX_REQUEST_OUT |
UX_REQUEST_TYPE_STANDARD |
UX_REQUEST_TARGET_DEVICE;
transfer_request -> ux_transfer_request_value =
   (USHORT) configuration -> ux_configuration_descriptor.bConfigurationValue;
transfer_request -> ux_transfer_request_index = 0;

/* Send request to HCD layer. */
status = ux_host_stack_transfer_request(transfer_request);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_create_printer_read

Read from printer interface

Prototype

```
UINT  ux_host_class_printer_read(UX_HOST_CLASS_PRINTER *printer, UCHAR
                      *data_pointer,
                      ULONG requested_length,
                      ULONG *actual_length)
```

Description

This function reads from the printer interface. The call is blocking and only returns when there is either an error or when the transfer is complete. A read is allowed only on bi-directional printers.

Input Parameters

printer	Pointer to the printer class instance.
data_pointer	Pointer to the buffer address of the data payload.
requested_length	Length to be received.
actual_length	Length actually received.

Return Values

UX_SUCCESS	(0x00)	The data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported because the printer is not bi-directional.
UX_TRANSFER_TIMEOUT	(0x5c)	Transfer timeout, reading incomplete.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_host_class_printer_read(printer, data_pointer,
                            requested_length, actual_length);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_printer_write

Write to printer interface

Prototype

```
UINT  ux_host_class_printer_write(UX_HOST_CLASS_PRINTER *printer,
                      UCHAR *data_pointer,
                      ULONG requested_length,
                      ULONG *actual_length)
```

Description

This function writes to the printer interface. The call is blocking and only returns when there is either an error or when the transfer is complete.

Input Parameters

printer	Pointer to printer class instance.
data_pointer	Pointer to buffer address of the data payload.
requested_length	Length to be sent.
actual_length	Length actually sent.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_TRANSFER_TIMEOUT	(0x5c)	Transfer timeout, writing incomplete.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_host_class_printer_write(printer, data_pointer,
                            requested_length, actual_length);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_printer_soft_reset

Perform a soft reset to the printer

Prototype

```
UINT      ux_host_class_printer_soft_reset(UX_HOST_CLASS_PRINTER *printer)
```

Description

This function performs a soft reset to the printer.

Input Parameters

printer	Pointer to the printer class instance.

Return Values

UX_SUCCESS	(0x00)	Reset was completed.
UX_TRANSFER_TIMEOUT	(0x05c)	Transer timeout; reset not completed.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_host_class_printer_soft_reset(printer);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_printer_status_get

Get printer status

Prototype

```
UINT ux_host_class_printer_status_get(UX_HOST_CLASS_PRINTER *printer,
                                      ULONG *printer_status)
```

Description

This function obtains the printer status. Printer status is similar to the LPT status (1284 standard).

Input Parameters

printer	Pointer to the printer class instance.
printer_status	Address of status to be returned.

Return Values

UX_SUCCESS	(0x00)	Reset was completed.
UX_MEMORY_INSUFFICIENT	(0x12)	Not enough memory to perform the operation.
UX_TRANSFER_TIMEOUT	(0x5c)	Transfer timeout; reset not completed

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_host_class_printer_status_get(printer, printer_status);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_audio_read

Read from audio interface

Prototype

```
UINT  ux_host_class_audio_read(UX_HOST_CLASS_AUDIO *audio,
                  UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST
                  *audio_transfer_request)
```

Description

This function reads from the audio interface. The call is non-blocking. The application must ensure that the appropriate alternate setting has been selected for the audio streaming interface.

Input Parameters

audio	Pointer to the audio class instance.
audio_transfer_request	Pointer to the audio transfer structure.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported

Example

```
/* The following example reads from the audio interface. */

audio_transfer_request.ux_host_class_audio_transfer_request_completion_function =
                                                tx_audio_transfer_completion_function;
audio_transfer_request.ux_host_class_audio_transfer_request_class_instance = audio;
audio_transfer_request.ux_host_class_audio_transfer_request_next_audio_audio_transfer_request =
                                                UX_NULL;
audio_transfer_request. ux_host_class_audio_transfer_request_data_pointer = audio_buffer;
audio_transfer_request.ux_host_class_audio_transfer_request_requested_length =
                                                requested_length;
audio_transfer_request.ux_host_class_audio_transfer_request_packet_length = AUDIO_FRAME_LENGTH;

status = ux_host_class_audio_read(audio, audio_transfer_request);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_audio_write

Read from audio interface

Prototype

```
UINT  ux_host_class_audio_write(UX_HOST_CLASS_AUDIO *audio,
       UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST *audio_transfer_request)
```

Description

This function writes to the audio interface. The call is non-blocking. The application must ensure that the appropriate alternate setting has been selected for the audio streaming interface.

Input Parameters

audio	Pointer to the audio class instance.
audio_transfer_request	Pointer to the audio transfer structure.

Return Values

UX_SUCCESS	(0x00)	The data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_AUDIO_WRONG_INTERFACE	(0x81)	Interface incorrect.

Example

```
UINT   status;

/* The following example writes to the audio interface */

audio_transfer_request.ux_host_class_audio_transfer_request_completion_function =
                                                    tx_audio_transfer_completion_function;
audio_transfer_request.ux_host_class_audio_transfer_request_class_instance = audio;
audio_transfer_request.ux_host_class_audio_transfer_request_next_audio_audio_transfer_
                                                    request = UX_NULL;
audio_transfer_request.ux_host_class_audio_transfer_request_data_pointer = audio_buffer;
audio_transfer_request.ux_host_class_audio_transfer_request_requested_length =
                                                    requested_length;
audio_transfer_request.ux_host_class_audio_transfer_request_packet_length =
                                                    AUDIO_FRAME_LENGTH;

status = ux_host_class_audio_write(audio, audio_transfer_request);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_audio_control_get

Get specific control from audio control interface

Prototype

```
UINT  ux_host_class_audio_control_get(UX_HOST_CLASS_AUDIO *audio,
                                      UX_HOST_CLASS_AUDIO_CONTROL *audio_control)
```

Description

This function reads a specific control from the audio control interface.

Input Parameters

audio	Pointer to the audio class instance.
audio_control	Pointer to the audio control structure.

Return Values

UX_SUCCESS	(0x00)	The data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_AUDIO_WRONG_INTERFACE	(0x81)	Interface incorrect.

Example

```
UINT   status;

/* The following example reads the volume control from a stereo USB
   speaker. */

UX_HOST_CLASS_AUDIO_CONTROLaudio_control;

audio_control. ux_host_class_audio_control_channel = 1;
audio_control. ux_host_class_audio_control =
   UX_HOST_CLASS_AUDIO_VOLUME_CONTROL;

status = ux_host_class_audio_control_get(audio, &audio_control);
/* If status equals UX_SUCCESS, the operation was successful. */

audio_control. ux_host_class_audio_control_channel = 2;
audio_control. ux_host_class_audio_control =
   UX_HOST_CLASS_AUDIO_VOLUME_CONTROL;

status = ux_host_class_audio_control_get(audio, &audio_control);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_audio_control_value_set

Set specific control to audio control interface

Prototype

```
UINT  ux_host_class_audio_control_value_set(UX_HOST_CLASS_AUDIO *audio,
                                            UX_HOST_CLASS_AUDIO_CONTROL *audio_control)
```

Description

This function sets a specific control to the audio control interface.

Input Parameters

audio	Pointer to the audio class instance.
audio_control	Pointer to the audio control structure.

Return Values

UX_SUCCESS	(0x00)	The data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_AUDIO_WRONG_INTERFACE	(0x81)	Interface incorrect.

Example

```
/* The following example sets the volume control of a stereo USB speaker. */

UX_HOST_CLASS_AUDIO_CONTROL   audio_control;
UINT   status;

audio_control. ux_host_class_audio_control_channel = 1;
audio_control. ux_host_class_audio_control = UX_HOST_CLASS_AUDIO_VOLUME_CONTROL;
audio_control. ux_host_class_audio_control_cur = 0xf000;

status = ux_host_class_audio_control_value_set(audio, &audio_control);
/* If status equals UX_SUCCESS, the operation was successful. */

current_volume = audio_control.audio_ control_cur;

audio_control. ux_host_class_audio_control_channel = 2;
audio_control. ux_host_class_audio_control = UX_HOST_CLASS_AUDIO_VOLUME_CONTROL;
audio_control. ux_host_class_audio_control_cur = 0xf000;

status = ux_host_class_audio_control_value_set(audio, &audio_control);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_audio_streaming_sampling_set

Set alternate setting interface of audio streaming interface

Prototype

```
UINT  ux_host_class_audio_streaming_sampling_set(UX_HOST_CLASS_AUDIO *audio,
                                UX_HOST_CLASS_AUDIO_SAMPLING *audio_sampling)
```

Description

This function sets the appropriate alternate setting interface of the audio streaming interface according to a specific sampling structure.

Input Parameters

audio	Pointer to the audio class instance.
audio_sampling	Pointer to the audio sampling structure.

Return Values

UX_SUCCESS	(0x00)	The data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_AUDIO_WRONG_INTERFACE	(0x81)	Interface incorrect.
UX_NO_ALTERNATE_SETTING	(0x5e)	No alternate setting for the sampling values.

Example

```
/* The following example sets the alternate setting interface of a
   stereo
   USB speaker. */

UX_HOST_CLASS_AUDIO_SAMPLING    audio_sampling;
UINT   status;

sampling. ux_host_class_audio_sampling_channels = 2;
sampling. ux_host_class_audio_sampling_frequency = AUDIO_FREQUENCY;
sampling. ux_host_class_audio_sampling_resolution = 16;

status = ux_host_class_audio_streaming_sampling_set(audio,
   &sampling);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_audio_streaming_sampling_get

Get sampling settings of audio streaming interface

Prototype

```
UINT  ux_host_class_audio_streaming_sampling_get(UX_HOST_CLASS_AUDIO *audio,
                  UX_HOST_CLASS_AUDIO_SAMPLING_CHARACTERISTICS *audio_sampling)
```

Description

This function gets, one by one, all the possible sampling settings available in each of the alternate settings of the audio streaming interface. The first time the function is used, all the fields in the calling structure pointer must be reset. The function will return a specific set of streaming values upon return unless the end of the alternate settings has been reached. When this function is reused, the previous sampling values will be used to find the next sampling values.

Input Parameters

audio	Pointer to the audio class instance.
audio_sampling	Pointer to the audio sampling structure.

Return Values

UX_SUCCESS	(0x00)	The data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_AUDIO_WRONG_INTERFACE	(0x81)	Interface incorrect.
UX_NO_ALTERNATE_SETTING	(0x5e)	No alternate setting for the sampling values.

Example

```
/* The following example gets the sampling values for the first
   alternate setting interface of a stereo USB speaker. */

UX_HOST_CLASS_AUDIO_SAMPLING_CHARACTERISTICS    audio_sampling;
UINT   status;

sampling.ux_host_class_audio_sampling_channels=0;
sampling.ux_host_class_audio_sampling_frequency_low=0;
sampling.ux_host_class_audio_sampling_frequency_high=0;
sampling.ux_host_class_audio_sampling_resolution=0;

status = ux_host_class_audio_streaming_sampling_get(audio,
   &sampling);

/* If status equals UX_SUCCESS, the operation was successful and
   information could be displayed as follows:

   printf("Number of channels %d, Resolution %d bits,
               frequency range %d-%d\n",
    sampling.audio_channels, sampling.audio_resolution,
    sampling.audio_frequency_low, sampling.audio_frequency_high);
*/
```

ux_host_class_hid_client_register

Register HID client to HID class

Prototype

```
UINT  ux_host_class_hid_client_register(UCHAR_PTR hid_client_name,
   UINT (*hid_client_handler) (
   struct UX_HOST_CLASS_HID_CLIENT_COMMAND_STRUCT *))
```

Description

This function is used to register an HID client to the HID class. The HID class needs to find a match between an HID device and HID client before requesting data from this device.

Input Parameters

hid_client_name	Pointer to the HID client name.
hid_client_handler	Pointer to the HID client handler.

Return Values

UX_SUCCESS	(0x00)	The data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_ALREADY_INSTALLED	(0x58)	This class already exists.

Example

```
UINT   status;

/* The following example illustrates how to register an HID client, in this
   case a USB mouse, to the HID class. */

status =
  ux_host_class_hid_client_register("ux_host_class_hid_client_mouse",
                                    ux_host_class_hid_mouse_entry);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_hid_report_callback_register

Register callback from HID class

Prototype

```
UINT  ux_host_class_hid_report_callback_register(UX_HOST_CLASS_HID *hid,
                                UX_HOST_CLASS_HID_REPORT_CALLBACK *call_back)
```

Description

This function is used to register a callback from the HID class to the HID client when a report is received.

Input Parameters

hid	Pointer to the HID class instance.
call_back	Pointer to the call_back structure.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_HID_REPORT_ERROR	(0x79)	Error in the report callback registration.

Example

```
UINT   status;

/* This example illustrates how to register an HID client, in this case a
   USB mouse, to the HID class. In this case, the HID client is asking the
   HID class to call the client for each usage received in the HID report.
   */

call_back.ux_host_class_hid_report_callback_id = 0;
call_back.ux_host_class_hid_report_callback_function =
                              ux_host_class_hid_mouse_callback;
call_back.ux_host_class_hid_report_callback_buffer = UX_NULL;
call_back.ux_host_class_hid_report_callback_flags =
                               UX_HOST_CLASS_HID_REPORT_INDIVIDUAL_USAGE;
call_back.ux_host_class_hid_report_callback_length = 0;

status = ux_host_class_hid_report_callback_register(hid, &call_back);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_hid_periodic_report_start

Start periodic endpoint for HID class instance

Prototype

```
UINT  ux_host_class_hid_periodic_report_start(UX_HOST_CLASS_HID *hid)
```

Description

This function is used to start the periodic (interrupt) endpoint for the instance of the HID class that is bound to this HID client. The HID class cannot start the periodic endpoint until the HID client is activated and, therefore, it is left to the HID client to start this endpoint to receive reports.

Input Parameters

hid	Pointer to the HID class instance.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_HID_PERIODIC_REPORT_ERROR	(0x7A)	Error in the periodic report.
UX_HOST_CLASS_INSTANCE_UNKNOWN	(0x5b)	HID class instance does not exist.

Example

```
UINT   status;

/* The following example illustrates how to start the periodic endpoint. */

status = ux_host_class_hid_periodic_report_start(hid);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_hid_periodic_report_stop

Stop periodic endpoint for HID class instance

Prototype

```
UINT  ux_host_class_hid_periodic_report_stop(UX_HOST_CLASS_HID *hid)
```

Description

This function is used to stop the periodic (interrupt) endpoint for the instance of the HID class that is bound to this HID client. The HID class cannot stop the periodic endpoint until the HID client is deactivated, all its resources freed, and, therefore, it is left to the HID client to stop this endpoint.

Input Parameters

hid	Pointer to the HID class instance.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_HID_PERIODIC_REPORT_ERROR	(0x7A)	Error in periodic report.
UX_HOST_CLASS_INSTANCE_UNKNOWN	(0x5b)	HID class instance does not exist.

Example

```
UINT   status;

/* The following example illustrates how to stop the periodic
   endpoint. */

status = ux_host_class_hid_periodic_report_stop(hid);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_hid_report_get

Get report from HID class instance

Prototype

```
UINT  ux_host_class_hid_report_get(UX_HOST_CLASS_HID *hid,
                      UX_HOST_CLASS_HID_CLIENT_REPORT *client_report)
```

Description

This function is used to receive a report directly from the device without relying on the periodic endpoint. This report is coming from the control endpoint but its treatment is the same as though it were coming on the periodic endpoint.

Input Parameters

hid	Pointer to the HID class instance.
client_report	Pointer to the HID client report.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_HID_REPORT_ERROR	(0x70)	Error in the periodic report.
UX_HOST_CLASS_INSTANCE_UNKNOWN	(0x5b)	HID class instance does not exist.
UX_BUFFER_OVERFLOW	(0x5d)	The buffer supplied is not big enough to accommodate the uncompressed report.

Example

```
UX_HOST_CLASS_HID_CLIENT_REPORT    input_report;
UINT   status;

/* The following example illustrates how to get a report. */

input_report.ux_host_class_hid_client_report = hid_report;
input_report.ux_host_class_hid_client_report_buffer = buffer;
input_report.ux_host_class_hid_client_report_length = length;
input_report.ux_host_class_hid_client_flags =
                      UX_HOST_CLASS_HID_REPORT_INDIVIDUAL_USAGE;

status = ux_host_class_hid_report_get(hid, &input_report);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_host_class_hid_report_set

Send a report

Prototype

```
UINT  ux_host_class_hid_report_set(UX_HOST_CLASS_HID *hid,
       UX_HOST_CLASS_HID_CLIENT_REPORT *client_report)
```

Description

This function is used to send a report directly to the device.

Input Parameters

hid	Pointer to the HID class instance.
client_report	Pointer to the HID client report.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_FUNCTION_NOT_SUPPORTED	(0x54)	Function not supported.
UX_HOST_CLASS_HID_REPORT_ERROR	(0x70)	Error in the periodic report.
UX_HOST_CLASS_INSTANCE_UNKNOWN(	0x5b)	HID class instance does not exist.
UX_BUFFER_OVERFLOW	(0x5d)	Buffer supplied is not big enough to accommodate the uncompressed report.

Example

```
/* The following example illustrates how to send a report. */

UX_HOST_CLASS_HID_CLIENT_REPORT    input_report;

input_report.ux_host_class_hid_client_report = hid_report;
input_report.ux_host_class_hid_client_report_buffer = buffer;
input_report.ux_host_class_hid_client_report_length = length;
input_report.ux_host_class_hid_client_report_flags =
               UX_HOST_CLASS_HID_REPORT_INDIVIDUAL_USAGE;

status = ux_host_class_hid_report_set(hid, &input_report);

/* If status equals UX_SUCCESS, the operation was successful. */
```

U S B X

CHAPTER 5

Device Stack for USBX

This chapter contains a description of the high performance USBX-embedded USB device stack from a functional perspective.

Execution Overview

USBX for the device is composed of several components. The following diagram illustrates the USBX device stack:

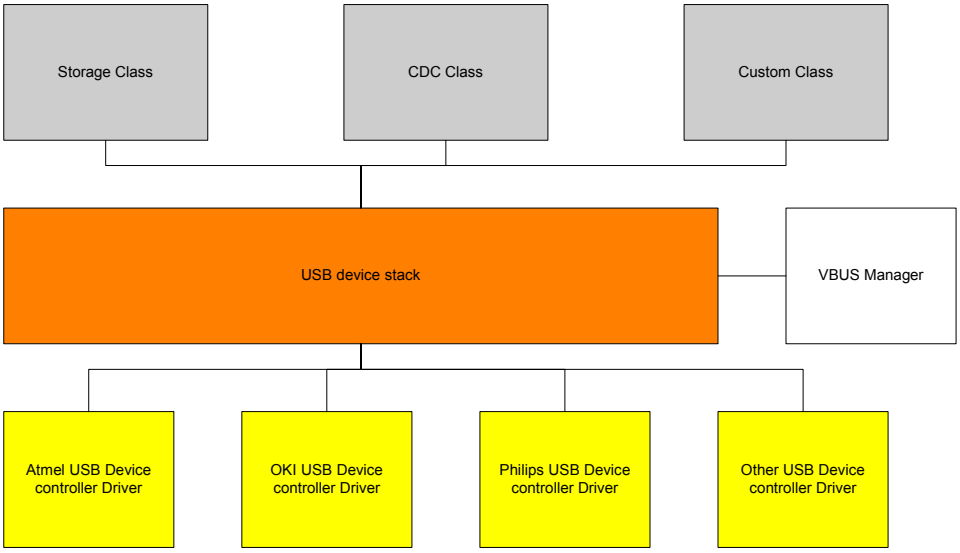


FIGURE 7. USBX Device Stack

Initialization

To activate USBX, the function *ux_system_initialize* must be called. This function initializes the memory resources of USBX.

To activate USBX device facilities, the function *ux_device_stack_initialize* must be called. This function will in turn initialize all the resources used by the USBX device stack such as ThreadX threads, mutexes, and semaphores.

The application initialization activates the USB device controller and one or more USB classes. Contrary to the USB host side, the device side can have only one USB controller driver running at any time. When the classes have been registered to the stack and the device controller initialization function has been called, the bus is active and the stack will reply to bus reset and host enumeration commands.

Application Interface Calls

There are two levels of APIs in USBX:

- USB device stack APIs
- USB device class APIs

Normally, a USBX application should not have to call any of the USB device stack APIs. Most applications will only access the USB Class APIs.

USB Device Stack APIs: The device stack APIs are responsible for the registration of USBX device components such as classes and the device framework.

USB Device Class APIs: The class APIs are very specific to each USB class. Most of the common APIs for USB classes provided services such as opening/ closing a device and reading from and writing to a device. The APIs are similar in nature to the host side.

Device Framework

The USB device side is responsible for the definition of the device framework. The device framework is divided into three categories, as described in the following sections.

Components of Device Framework

The definition of each component of the device framework is related to the nature of the device and the resources utilized by the device. Following are the main categories.

- Device Descriptor
- Configuration Descriptor
- Interface Descriptor
- Endpoint Descriptor

USBX supports device component definition for both high and full speed (low speed being treated the same way as full speed). This allows the device to operate differently when connected to a high speed or full speed host. The typical differences are the size of each endpoint and the power consumed by the device.

The definition of the device component takes the form of a byte string that follows the USB specification. The definition is contiguous and the order in which the framework is represented in memory will be the same as the one returned to the host during enumeration.

Following is an example of a device framework for a high speed USB Flash Disk:

```
#define DEVICE_FRAMEWORK_LENGTH_HIGH_SPEED 60
UCHAR device_framework_high_speed[] = {

        /* Device descriptor */
                0x12, 0x01, 0x00, 0x02, 0x00, 0x00, 0x00, 0x40,
                0x0a, 0x07, 0x25, 0x40, 0x01, 0x00, 0x01, 0x02,
                0x03, 0x01,

        /* Device qualifier descriptor */
                0x0a, 0x06, 0x00, 0x02, 0x00, 0x00, 0x00, 0x40,
                0x01, 0x00,

        /* Configuration descriptor */
                0x09, 0x02, 0x20, 0x00, 0x01, 0x01, 0x00, 0xc0,
                0x32,

        /* Interface descriptor */
                0x09, 0x04, 0x00, 0x00, 0x02, 0x08, 0x06, 0x50,
                0x00,

        /* Endpoint descriptor (Bulk Out) */
                0x07, 0x05, 0x01, 0x02, 0x00, 0x02, 0x00,

        /* Endpoint descriptor (Bulk In) */
                0x07, 0x05, 0x82, 0x02, 0x00, 0x02, 0x00
        };
```

Strings of Device Framework

Strings are optional in a device. Their purpose is to let the USB host know about the manufacturer of the device, the product name, and the revision number through Unicode strings.

The main strings are indexes embedded in the device descriptors. Additional strings indexes can be embedded into individual interfaces.

Assuming the device framework above has three string indexes embedded into the device descriptor, the string framework definition could look like this:

```
/* String Device Framework:
     Byte 0 and 1: Word containing the language ID: 0x0904 for US
     Byte 2     : Byte containing the index of the descriptor
     Byte 3     : Byte containing the length of the descriptor string
*/

#define STRING_FRAMEWORK_LENGTH 38
UCHAR string_framework[] = {

/* Manufacturer string descriptor: Index 1 */
        0x09, 0x04, 0x01, 0x0c,
        0x45, 0x78, 0x70, 0x72, 0x65, 0x73, 0x20, 0x4c,
        0x6f, 0x67, 0x69, 0x63,

/* Product string descriptor: Index 2 */
        0x09, 0x04, 0x02, 0x0c,
        0x4D, 0x4C, 0x36, 0x39, 0x36, 0x35, 0x30, 0x30,
        0x20, 0x53, 0x44, 0x4B,

/* Serial Number string descriptor: Index 3 */
        0x09, 0x04, 0x03, 0x04,
        0x30, 0x30, 0x30, 0x31
};
```

If different strings have to be used for each speed, different indexes must be used as the indexes are speed agnostic.

The encoding of the string is UNICODE-based. For more information on the UNICODE encoding standard refer to the following publication:

The Unicode Standard, Worldwide Character Encoding, *Version 1., Volumes 1 and 2, The Unicode Consortium,* Addison-Wesley Publishing Company, Reading MA.

Languages Supported by Device for each String

USBX has the ability to support multiple languages, although English is the default. The definition of each language for the string descriptors is in the form of an array of languages definition defined as follows:

```
#define LANGUAGE_ID_FRAMEWORK_LENGTH 2
UCHAR language_id_framework[] = {

        /* English. */
             0x09, 0x04
        };
```

To support additional languages, simply add the language code double-byte definition after the default English code. The language code has been defined by Microsoft in the document:

> *Developing International Software for Windows 95 and Windows NT*, Nadine Kano, Microsoft Press, Redmond WA

VBUS Manager

In most USB device designs, VBUS is not part of the USB Device core but rather connected to an external GPIO, which monitors the line signal. As a result, VBUS has to be managed separately from the device controller driver.

It is up to the application to provide the device controller with the address of the VBUS IO. VBUS must be initialized prior to the device controller initialization.

Depending on the platform specification for monitoring VBUS, it is possible to let the controller driver handle VBUS signals after the VBUS IO is initialized or if this is not possible, the application has to provide the code for handling VBUS.

If the application wishes to handle VBUS by itself, its only requirement is to call the function

```
ux_device_stack_disconnect()
```

when it detects that a device has been extracted. It is not necessary to inform the controller when a device is inserted because the controller will wake up when the BUS RESET assert/deassert signal is detected.

CHAPTER 6

Description of USBX Device Stack Services

This chapter contains a description of all USBX services in alphabetic order. Service names are designed so all similar services are grouped together.

U S B X

ux_device_stack_alternate_setting_get

Get current alternate setting for interface value

Prototype

```
UINT ux_device_stack_alternate_setting_get(ULONG interface_value)
```

Description

This function is used by the USB host to obtain the current alternate setting for a specific interface value. It is called by the controller driver when a GET_INTERFACE request is received.

Input Parameters

interface_value	Interface value for which current alternate setting is queried.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_ERROR	(0xFF)	Wrong interface value.

Example

```
ULONG   interface_value;
UINT    status;

/* The following example illustrates this service. */

status = ux_device_stack_alternate_setting_get(interface_value);

/* If status equals UX_SUCCESS, the operation was successful. */
/* We need to abort transactions on the bulk out pipe. */
status = ux_host_stack_endpoint_transfer_abort
           (printer -> printer_bulk_out_endpoint);

/* If status equals UX_SUCCESS, the operation was successful */
```

ux_device_stack_alternate_setting_set

Set current alternate setting for interface value

Prototype

```
UINT ux_device_stack_alternate_setting_set(ULONG interface_value,
                                           ULONG alternate_setting_value)
```

Description

This function is used by the USB host to set the current alternate setting for a specific interface value. It is called by the controller driver when a SET_INTERFACE request is received. When the SET_INTERFACE is completed, the values of the alternate settings are applied to the class.

Input Parameters

interface_value	Interface value for which the current alternate setting is set.
alternate_setting_value	New alternate setting value.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_INTERFACE_HANDLE_UNKNOWN	(0x52)	No interface attached.
UX_ERROR	(0xFF)	Wrong interface value.

Example

```
ULONG   interface_value;
ULONG   alternate_setting_value;

/* The following example illustrates this service. */

status = ux_device_stack_alternate_setting_set(interface_value,
                                               alternate_setting_value);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_class_register

Register new USB device class

Prototype

```
UINT  ux_device_stack_class_register(UCHAR *class_name,
          UINT (*class_entry_function)
                    (struct UX_SLAVE_CLASS_COMMAND_STRUCT *),
          VOID (*class_thread_function)(ULONG),
          ULONG interface_number,
          VOID *parameter)
```

Description

This function is used by the application to register a new USB device class. This registration starts a class container and not an instance of the class. A class should have an active thread and be attached to a specific interface.

Some classes expect a parameter or parameter list. For instance, the device storage class would expect the geometry of the storage device it is trying to emulate. The parameter field is therefore dependent on the class requirement and can be a value or a pointer to a structure filled with the class values.

Input Parameters

class_entry_function	Entry function of class.
class_thread_function	Active thread function of class.
interface_number	Interface number this class is attached to.
parameter	Pointer to a class specific parameter list.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_MEMORY_INSUFFICIENT	(0x52)	Not enough memory.
UX_THREAD_ERROR	(0xFF)	Cannot create a class thread.

Example

```
UINT   status;

/* The following example illustrates this service. */

/* Initialize the device storage class. The class is connected with
   interface 1 */

status =
    ux_device_stack_class_register(_ux_system_slave_class_storage_name,
                _ux_device_class_storage_entry, 1,  (VOID *)&parameter);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_configuration_get

Get current configuration

Prototype

```
UINT ux_device_stack_configuration_get(VOID)
```

Description

This function is used by the host to obtain the current configuration running in the device.

Input Parameters

No input parameters.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_device_stack_configuration_get();

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_configuration_set

Set current configuration

Prototype

```
UINT ux_device_stack_configuration_set(ULONG configuration_value)
```

Description

This function is used by the host to set the current configuration running in the device. Upon reception of this command, the USB device stack will activate the alternate setting 0 of each interface connected to this configuration.

Input Parameters

configuration_value	Configuration value selected by host.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.

Example

```
ULONG   configuration_value;
UINT    status;

/* The following example illustrates this service. */

status = ux_device_stack_configuration_set(configuration_value);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_descriptor_send

Send descriptor to host

Prototype

```
UINT ux_device_stack_descriptor_send(ULONG descriptor_type,
                                     ULONG request_index, ULONG host_length)
```

Description

This function is used by the device side to return a descriptor to the host. This descriptor can be a device descriptor, a configuration descriptor, or a string descriptor.

Input Parameters

descriptor_type	Nature of descriptor: UX_DEVICE_DESCRIPTOR_ITEM UX_CONFIGURATION_DESCRIPTOR_ITEM UX_STRING_DESCRIPTOR_ITEM UX_DEVICE_QUALIFIER_DESCRIPTOR_ITEM UX_OTHER_SPEED_DESCRIPTOR_ITEM
request_index	Index of descriptor.
host_length	Length required by host.

Return Values

UX_SUCCESS	(0x00)	Data transfer was completed.
UX_ERROR	(0xFF)	Transfer was not completed.

Example

```
ULONG   descriptor_type;
ULONG   request_index;
ULONG   host_length;
UINT    status;

/* The following example illustrates this service. */

status = ux_device_stack_configuration_send(descriptor_type,
                                            request_index, host_length);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_disconnect

Disconnect device stack

Prototype

```
UINT ux_device_stack_disconnect(VOID)
```

Description

VBUS manager calls this function when there is a device disconnection. The device stack will inform all classes registered to this device and will thereafter release all the device resources.

Input Parameters

No input parameters.

Return Values

UX_SUCCESS	(0x00)	Device was disconnected.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_device_stack_disconnected();

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_endpoint_stall

Request endpoint stall condition

Prototype

```
UINT ux_device_stack_endpoint_stall(UX_SLAVE_ENDPOINT *endpoint)
```

Description

This function is called by the USB device class when an endpoint should return a stall condition to the host.

Input Parameters

endpoint	Endpoint on which the stall condition is requested.

Return Values

UX_SUCCESS	(0x00)	Operation was successful.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_device_stack_endpoint_stall(endpoint);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_host_wakeup

Wake up host

Prototype

```
UINT ux_device_stack_host_wakeup(VOID)
```

Description

This function is called when the device wants to wake up the host. This command is only valid when the device is in suspend mode. It is up to the device application to decide when it wants to wake up the USB host. For instance, a USB modem can wake up a host when it detects a RING signal on the telephone line.

Input Parameters

No input parameters.

Return Values

UX_SUCCESS	(0x00)	Call was successful.
UX_ERROR	(0xFF)	Call failed; device was probably not in the suspended mode.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_device_stack_host_wakeup();

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_initialize

Initialize USB device stack

Prototype

```
UINT ux_device_stack_initialize(UCHAR_PTR device_framework_high_speed,
                      ULONG device_framework_length_high_speed,
                      UCHAR_PTR device_framework_full_speed,
                      ULONG device_framework_length_full_speed,
                      UCHAR_PTR string_framework,
                      ULONG string_framework_length,
                      UCHAR_PTR language_id_framework,
                      ULONG language_id_framework_length),
                      UINT (*ux_system_slave_change_function)(ULONG)))
```

Description

This function is called by the application to initialize the USB device stack. It does not initialize any classes or any controllers. This should be done with separate function calls. This call mainly provides the stack with the device framework for the USB function. It supports both high and full speeds with the possibility to have completely separate device framework for each speed. String framework and multiple languages are supported.

Input Parameters

device_framework_high_speed
Pointer to the high speed framework.

device_framework_length_high_speed
Length of the high speed framework.

device_framework_full_speed
Pointer to the full speed framework.

device_framework_length_full_speed
Length of the full speed framework.

string_framework Pointer to string framework.

string_framework_length Length of string framework.

language_id_framework Pointer to string language framework.

language_id_framework_length
Length of the string language framework.

ux_system_slave_change_function
Function to be called when the device state changes.

Return Values

UX_SUCCESS	(0x00)	This operation was successful.
UX_MEMORY_INSUFFICIENT	(0x12)	Not enough memory to initialize the stack.

Example

```
/* Example of a device framework */

#define DEVICE_FRAMEWORK_LENGTH_FULL_SPEED 50
UCHAR device_framework_full_speed[] = {

      /* Device descriptor */
      0x12, 0x01, 0x10, 0x01, 0x00, 0x00, 0x00, 0x08,
      0xec, 0x08, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x01,

      /* Configuration descriptor */
      0x09, 0x02, 0x20, 0x00, 0x01, 0x01, 0x00, 0xc0,
      0x32,

      /* Interface descriptor */
      0x09, 0x04, 0x00, 0x00, 0x02, 0x08, 0x06, 0x50,
      0x00,

      /* Endpoint descriptor (Bulk Out) */
      0x07, 0x05, 0x01, 0x02, 0x40, 0x00, 0x00,

      /* Endpoint descriptor (Bulk In) */
      0x07, 0x05, 0x82, 0x02, 0x40, 0x00, 0x00
      };

#define DEVICE_FRAMEWORK_LENGTH_HIGH_SPEED 60
UCHAR device_framework_high_speed[] = {

      /* Device descriptor */
      0x12, 0x01, 0x00, 0x02, 0x00, 0x00, 0x00, 0x40,
      0x0a, 0x07, 0x25, 0x40, 0x01, 0x00, 0x01, 0x02,
      0x03, 0x01,

      /* Device qualifier descriptor */
                      0x0a, 0x06, 0x00, 0x02, 0x00, 0x00, 0x00, 0x40,
                      0x01, 0x00,

      /* Configuration descriptor */
      0x09, 0x02, 0x20, 0x00, 0x01, 0x01, 0x00, 0xc0,
      0x32,
```

```
      /* Interface descriptor */
      0x09, 0x04, 0x00, 0x00, 0x02, 0x08, 0x06, 0x50,
      0x00,

      /* Endpoint descriptor (Bulk Out) */
      0x07, 0x05, 0x01, 0x02, 0x00, 0x02, 0x00,

      /* Endpoint descriptor (Bulk In) */
      0x07, 0x05, 0x82, 0x02, 0x00, 0x02, 0x00
      };


      /* String Device Framework:
     Byte 0 and 1: Word containing the language ID: 0x0904 for US
     Byte 2      : Byte containing the index of the descriptor
     Byte 3      : Byte containing the length of the descriptor string
     */

#define STRING_FRAMEWORK_LENGTH 38
UCHAR string_framework[] = {

      /* Manufacturer string descriptor: Index 1 */
      0x09, 0x04, 0x01, 0x0c,
      0x45, 0x78, 0x70, 0x72,0x65, 0x73, 0x20, 0x4c,
      0x6f, 0x67, 0x69, 0x63,

      /* Product string descriptor: Index 2 */
      0x09, 0x04, 0x02, 0x0c,
      0x4D, 0x4C, 0x36, 0x39, 0x36, 0x35, 0x30, 0x30,
      0x20, 0x53, 0x44, 0x4B,

      /* Serial Number string descriptor: Index 3 */
      0x09, 0x04, 0x03, 0x04,
      0x30, 0x30, 0x30, 0x31
      };

      /* Multiple languages are supported on the device, to add
         a language besides English, the Unicode language code must
         be appended to the language_id_framework array and the length
         adjusted accordingly. */

#define LANGUAGE_ID_FRAMEWORK_LENGTH 2
UCHAR language_id_framework[] = {

/* English. */
      0x09, 0x04
};
```

The application can request a call back when the controller changes its state. The two main states for the controller are:

UX_DEVICE_SUSPENDED

UX_DEVICE_RESUMED

If the application does not need Suspend/Resume signals, it would supply a UX_NULL function.

```
UINT   status;

/* The code below is required for installing the device portion of USBX.
   There is no call back for device status change in this example. */

status = ux_device_stack_initialize(&device_framework_high_speed,
                        DEVICE_FRAMEWORK_LENGTH_HIGH_SPEED,
                        &device_framework_full_speed,
                        DEVICE_FRAMEWORK_LENGTH_FULL_SPEED,
                        &string_framework,
                        STRING_FRAMEWORK_LENGTH,
                        &language_id_framework,
                        LANGUAGE_ID_FRAMEWORK_LENGTH,
                        UX_NULL);

/* If status equals UX_SUCCESS, initialization was successful. */
```

ux_device_stack_interface_delete

Delete stack interface

Prototype

```
UINT ux_device_stack_interface_delete(UX_SLAVE_INTERFACE *interface)
```

Description

This function is called when an interface should be removed. An interface is removed either when a device is extracted, or following a bus reset, or when there is a new alternate setting.

Input Parameters

interface	Pointer to the interface to remove.

Return Values

UX_SUCCESS	(0x00)	Operation was successful.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_device_stack_interface_delete(interface);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_interface_get

Get current interface value

Prototype

```
UINT ux_device_stack_interface_get(UINT interface_value)
```

Description

This function is called when the host queries the current interface. The device returns the current interface value.

Input Parameters

interface_value	Interface value to return.

Return Values

UX_SUCCESS	(0x00)	Operation was successful.
UX_ERROR	(0xFF)	No interface exists.

Example

```
ULONG   interface_value;
UINT    status;

/* The following example illustrates this service. */

status = ux_device_stack_interface_delete(interface_value);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_interface_set

Change alternate setting of interface

Prototype

```
UINT ux_device_stack_interface_set(UCHAR_PTR device_framework,
                                   ULONG device_framework_length,
                                   ULONG alternate_setting_value)
```

Description

This function is called when the host requests a change of the alternate setting for the interface.

Input Parameters

device_framework	Address of the device framework for this interface.
device_framework_length	Length of the device framework.
alternate_setting_value	Alternate setting value to be used by this interface.

Return Values

UX_SUCCESS	(0x00)	Operation was successful.
UX_ERROR	(0xFF)	No interface exists.

Example

```
UCHAR_PTR  device_framework
ULONG      device_framework_length;
ULONG      alternate_setting_value;
UINT       status;

/* The following example illustrates this service. */

status = ux_device_stack_interface_set(device_framework,
               device_framework_length,
               alternate_setting_value);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_interface_start

Start search for class to own an interface instance

Prototype

```
UINT ux_device_stack_interface_start(UX_SLAVE_INTERFACE *interface)
```

Description

This function is called when an interface has been selected by the host and the device stack needs to search for a device class to own this interface instance.

Input Parameters

interface	Pointer to the interface created.

Return Values

UX_SUCCESS	(0x00)	Operation was successful.
UX_NO_CLASS_MATCH	(0x57)	No class exists for this interface.

Example

```
UINT   status;

/* The following example illustrates this service. */

status = ux_device_stack_interface_start(interface);

/* If status equals UX_SUCCESS, the operation was successful. */
```

ux_device_stack_transfer_request

Request to transfer data to host

Prototype

```
UINT ux_device_stack_transfer_request(UX_SLAVE_TRANSFER *transfer_request,
                                      ULONG slave_length,
                                      ULONG host_length)
```

Description

This function is called when a class or the stack wants to transfer data to the host. The host always polls the device but the device can prepare data in advance.

Input Parameters

transfer_request	Pointer to the transfer request.
slave_length	Length the device wants to return.
host_length	Length the host has requested.

Return Values

UX_SUCCESS	(0x00)	Operation was successful.
UX_ERROR	(0xFF)	Transport error.

Example

```
UINT   status;
/* The following example illustrates how to transfer more data
   than an application requests. */

while(total_length)
{
   /* How much can we send in this transfer? */
   if (total_length > UX_SLAVE_CLASS_STORAGE_BUFFER_SIZE)
      transfer_length = UX_SLAVE_CLASS_STORAGE_BUFFER_SIZE;
   else
      transfer_length = total_length;

   /* Copy the Storage Buffer into the transfer request memory. */
   _ux_utility_memory_copy(transfer_request ->
               ux_slave_transfer_request_data_pointer,
               media_memory, transfer_length);

   /* Send the data payload back to the caller. */
   status = ux_device_transfer_request(transfer_request,
                                transfer_length, transfer_length);
/* If status equals UX_SUCCESS, the operation was successful. */

   /* Update the buffer address.  */
   media_memory += transfer_length;

   /* Update the length to remain. */
   total_length -= transfer_length;
}
```

ux_device_stack_transfer_request_abort

Cancel a transfer request

Prototype

```
UINT ux_device_stack_transfer_abort(UX_SLAVE_TRANSFER *transfer_request,
                                    ULONG completion_code)
```

Description

This function is called when an application needs to cancel a transfer request or when the stack needs to abort a transfer request associated with an endpoint.

Input Parameters

transfer_request	Pointer to the transfer request.
completion_code	Error code to be returned to the class waiting for this transfer request to complete.

Return Values

UX_SUCCESS	(0x00)	Operation was successful.

Example

```
UINT   status;

/* The following example illustrates how to abort a transfer when
   a bus reset has been detected on the bus. */

status = ux_device_stack_transfer_abort(transfer_request,UX_TRANSFER_BUS_RESET);

/* If status equals UX_SUCCESS, the operation was successful. */
```

APPENDIX A

USBX Classes

This chapter contains a description of USBX device storage and DPUMP classes:

USB Device Storage Class

The USB device storage class allows for a storage device embedded in the system to be made visible to a USB host.

The USB device storage class does not by itself provide a storage solution. It merely accepts and interprets SCSI requests coming from the host. When one of these requests is a read or a write command, it will invoke a pre-defined call back to a real storage device handler, such as an ATA device driver or a Flash device driver.

When initializing the device storage class, a pointer structure is given to the class that contains all the information necessary. An example is given below.

```
/* Store the number of LUN in this device storage instance : single LUN.  */
   storage_parameter.ux_slave_class_storage_parameter_number_lun = 1;


/* Initialize the storage class parameters for reading/writing to the Flash
Disk.*/

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_storag
e_media_last_lba= 0x1e6bfe;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_stora
ge_media_block_length=512;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_stora
ge_media_type=0;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_stora
ge_media_removable_flag
       =  0x80;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_stora
ge_media_read
       =  tx_demo_thread_flash_media_read;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_stora
ge_media_write
       =  tx_demo_thread_flash_media_write;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class_stora
ge_media_status
       =  tx_demo_thread_flash_media_status;
```

```
 /* Initialize the device storage class. The class is connected with interface 0
*/
     status =
_ux_device_stack_class_register(_ux_system_slave_class_storage_name,
                                                _ux_device_class_storage_entry,
                                               _ux_device_class_storage_thread,0,
                                                (VOID *)&storage_parameter);
```

In this example, the drive's last block address or LBA is given as well as the logical sector size. The LBA is the number of sectors available in the media –1. The block length is set to 512 in regular storage media. It can be set to 2048 for optical drives.

The application needs to pass three callback function pointers to allow the storage class to read, write and obtain status for the media.

The prototypes for the read and write functions are:

```
UINT    media_read(UCHAR_PTR data_pointer, ULONG number_blocks, ULONG lba);
UINT    media_write(UCHAR_PTR data_pointer, ULONG number_blocks, ULONG lba);
```

where

data_pointer	is the address of the buffer to be used for reading or writing
number_blocks	is the number of sectors to read/write
lba	is the sector address to read.

The return value can have either the value UX_SUCCESS or UX_ERROR indicating a successful or unsuccessful operation. These operations do not need to return any other error codes. If there is an error in any operation, the storage class will invoke the status call back function.

This function has the following prototype:

```
ULONG    tx_demo_thread_media_status(ULONG media_id);
```

The calling parameter **media_id** is not currently used and should always be 0. In the future it may be used to distinguish multiple storage devices or storage devices with multiple SCSI LUNs. This version of the storage class does not support multiple instances of the storage class or storage devices with multiple SCSI LUNs.

The return value is a SCSI error code that can have the following format:

Bits 0-7	Sense_key
Bits 8-15	Additional Sense Code
Bits 16-23	Additional Sense Code Qualifier

The following table provides the possible Sense/ ASC/ASCQ combinations.

Sense Key	ASC	ASCQ	Description
00	00	00	NO SENSE
01	17	01	RECOVERED DATA WITH RETRIES
01	18	00	RECOVERED DATA WITH ECC
02	04	01	LOGICAL DRIVE NOT READY - BECOMING READY
02	04	02	LOGICAL DRIVE NOT READY - INITIALIZATION REQUIRED
02	04	04	LOGICAL UNIT NOT READY - FORMAT IN PROGRESS
02	04	FF	LOGICAL DRIVE NOT READY - DEVICE IS BUSY
02	06	00	NO REFERENCE POSITION FOUND
02	08	00	LOGICAL UNIT COMMUNICATION FAILURE
02	08	01	LOGICAL UNIT COMMUNICATION TIME-OUT
02	08	80	LOGICAL UNIT COMMUNICATION OVERRUN
02	3A	00	MEDIUM NOT PRESENT
02	54	00	USB TO HOST SYSTEM INTERFACE FAILURE
02	80	00	INSUFFICIENT RESOURCES
02	FF	FF	UNKNOWN ERROR
03	02	00	NO SEEK COMPLETE
03	03	00	WRITE FAULT
03	10	00	ID CRC ERROR
03	11	00	UNRECOVERED READ ERROR
03	12	00	ADDRESS MARK NOT FOUND FOR ID FIELD
03	13	00	ADDRESS MARK NOT FOUND FOR DATA FIELD

03	14	00	RECORDED ENTITY NOT FOUND
03	30	01	CANNOT READ MEDIUM - UNKNOWN FORMAT
03	31	01	FORMAT COMMAND FAILED
04	40	NN	DIAGNOSTIC FAILURE ON COMPONENT NN (80H-FFH)
05	1A	00	PARAMETER LIST LENGTH ERROR
05	20	00	INVALID COMMAND OPERATION CODE
05	21	00	LOGICAL BLOCK ADDRESS OUT OF RANGE
05	24	00	INVALID FIELD IN COMMAND PACKET
05	25	00	LOGICAL UNIT NOT SUPPORTED
05	26	00	INVALID FIELD IN PARAMETER LIST
05	26	01	PARAMETER NOT SUPPORTED
05	26	02	PARAMETER VALUE INVALID
05	39	00	SAVING PARAMETERS NOT SUPPORT
06	28	00	NOT READY TO READY TRANSITION – MEDIA CHANGED
06	29	00	POWER ON RESET OR BUS DEVICE RESET OCCURRED
06	2F	00	COMMANDS CLEARED BY ANOTHER INITIATOR
07	27	00	WRITE PROTECTED MEDIA
0B	4E	00	OVERLAPPED COMMAND ATTEMPTED

Multiple SCSI LUN

The USBX device storage class supports multiple LUNs. It is therefore possible to create a storage device that acts as a CD-ROM and a Flash disk at the same time. In such a case, the initialization is slightly different.

Here are examples for a Flash Disk and CD-ROM:

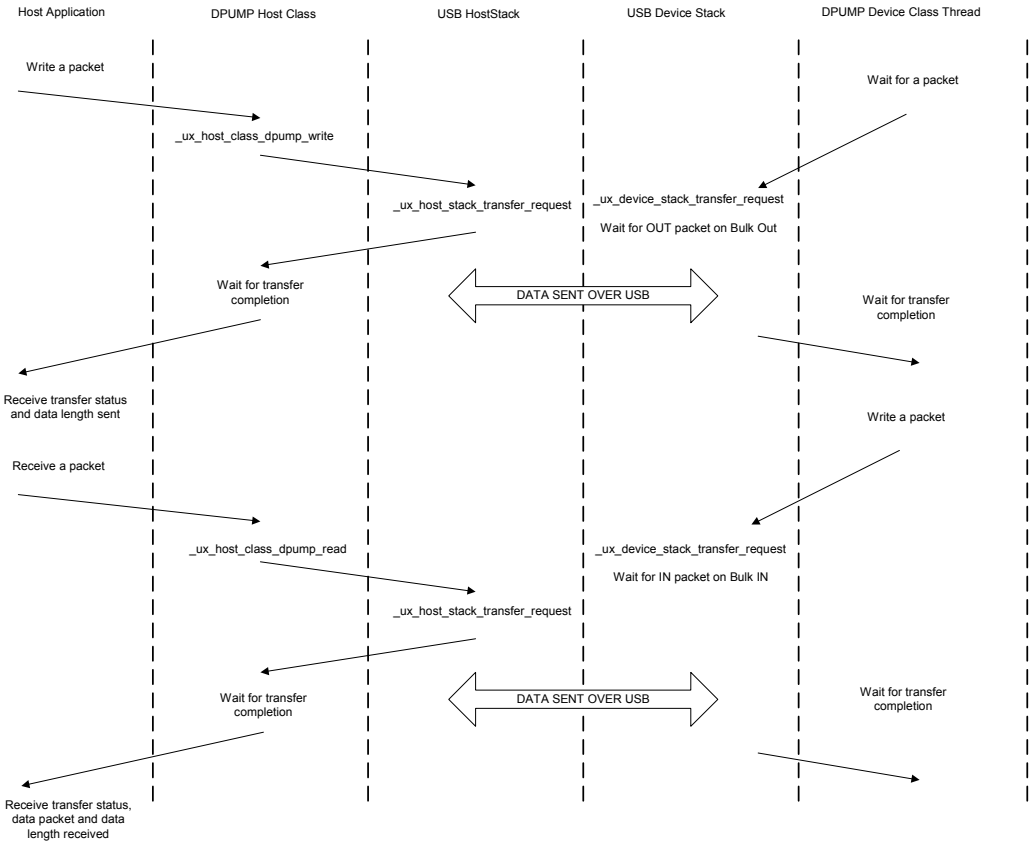
```
 /* Store the number of LUN in this device storage instance.  */
    storage_parameter.ux_slave_class_storage_parameter_number_lun = 2;


/* Initialize the storage class parameters for reading/writing to the
Flash Disk.  */

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class
_storage_media_last_lba = 0x7bbff;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class
_storage_media_block_length =  512;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class
_storage_media_type =  0;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class
_storage_media_removable_flag =  0x80;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class
_storage_media_read =  tx_demo_thread_flash_media_read;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class
_storage_media_write =  tx_demo_thread_flash_media_write;

storage_parameter.ux_slave_class_storage_parameter_lun[0].ux_slave_class
_storage_media_status =  tx_demo_thread_flash_media_status;


/* Initialize the storage class LUN parameters for reading/writing to the
CD-ROM.  */

storage_parameter.ux_slave_class_storage_parameter_lun[1].ux_slave_class
_storage_media_last_lba = 0x04caaf;

storage_parameter.ux_slave_class_storage_parameter_lun[1].ux_slave_class
_storage_media_block_length =  2048;

storage_parameter.ux_slave_class_storage_parameter_lun[1].ux_slave_class
_storage_media_type =  5;

storage_parameter.ux_slave_class_storage_parameter_lun[1].ux_slave_class
_storage_media_removable_flag =  0x80;
```

```
storage_parameter.ux_slave_class_storage_parameter_lun[1].ux_slave_class_
storage_media_read =  tx_demo_thread_cdrom_media_read;

storage_parameter.ux_slave_class_storage_parameter_lun[1].ux_slave_class_
storage_media_write =  tx_demo_thread_cdrom_media_write;

storage_parameter.ux_slave_class_storage_parameter_lun[1].ux_slave_class_
storage_media_status =  tx_demo_thread_cdrom_media_status;


/* Initialize the device storage class for a Flash disk and CD-ROM.
The class is connected with interface 0 */
status =
_ux_device_stack_class_register(_ux_system_slave_class_storage_name,

_ux_device_class_storage_entry,

_ux_device_class_storage_thread,0,
                                                (VOID *)&storage_parameter);
```

USB DPUMP Classes

USBX contains a DPUMP class for the host and device side. This class is not a standard class per se but rather an example that illustrates how to create a simple device by using two bulk pipes and sending data back and forth on these two pipes. The DPUMP class could be used to start a custom class or for legacy RS232 devices.

USB DPUMP flowchart:



USB DPUMP Host Class

The host side of the DPUMP Class has two functions: one for sending data and one for receiving data:

_ux_host_class_dpump_write

_ux_host_class_dpump_read

Both functions are blocking to make the DPUMP application easier. If it is necessary to have both pipes (IN and OUT)

running at the same time, the application will be required to create a transmit thread and a receive thread.

The prototype for the writing function is as follows:

```
UINT
_ux_host_class_dpump_write(UX_HOST_CLASS_DPUMP
*dpump, UCHAR * data_pointer,
ULONG requested_length, ULONG *actual_length)
```

Where :

dpump	is the instance of the class
data_pointer	is the pointer to the buffer to be sent
requested_length	is the length to send
actual_length	is the length sent after completion of the transfer, either successfully or partially.

The prototype for the receiving function is the same :

```
UINT
_ux_host_class_dpump_read(UX_HOST_CLASS_DPUMP
*dpump, UCHAR *data_pointer,
ULONG requested_length, ULONG *actual_length)
```

Here is an example of the host DPUMP class where an application writes a packet to the device side and receives the same packet on the reception:

```
/* We start with a 'A' in buffer.  */
   current_char = 'A';

   while(1)
   {
       /* Initialize the write buffer. */
       _ux_utility_memory_set(out_buffer, current_char,
                              UX_HOST_CLASS_DPUMP_PACKET_SIZE);

       /* Increment the character in buffer.  */
       current_char++;


       /* Check for upper alphabet limit.  */
       if (current_char > 'Z')
           current_char =  'A';

       /* Write to the Data Pump Bulk out endpoint.  */
       status = _ux_host_class_dpump_write (dpump, out_buffer,
                                UX_HOST_CLASS_DPUMP_PACKET_SIZE,
  &actual_length);

      /* Verify that the status and the amount of data is correct.
  */
       if ((status == UX_SUCCESS) && actual_length ==
  UX_HOST_CLASS_DPUMP_PACKET_SIZE)

       {
           /* Read to the Data Pump Bulk out endpoint.  */
           status = _ux_host_class_dpump_read (dpump, in_buffer,
                                 UX_HOST_CLASS_DPUMP_PACKET_SIZE,
  &actual_length);

       }
```

USB DPUMP Device Class

The device DPUMP class uses a thread that is started upon connection to the USB host. The thread waits for a packet coming on the Bulk Out endpoint. When a packet is received, it copies the content to the Bulk In endpoint buffer and posts a transaction on this endpoint, waiting for the host to issue a request to read from this endpoint. This provides a loopback mechanism between the Bulk Out and Bulk In endpoints.

APPENDIX **B**

USBX Constants

This appendix includes an alphabetic list of all USBX constants and their values.

UX_ALIGN_16	0x0f
UX_ALIGN_2048	0x7ff
UX_ALIGN_256	0xff
UX_ALIGN_32	0x1f
UX_ALIGN_4096	0xfff
UX_ALIGN_512	0x1ff
UX_ALIGN_512	0x1ff
UX_ALIGN_64	0x3f
UX_ALIGN_MIN	0x0f
UX_BUFFER_OVERFLOW	0x5d
UX_BULK_ENDPOINT	2
UX_BULK_ENDPOINT_IN	0x82
UX_BULK_ENDPOINT_OUT	0x02
UX_CACHE_SAFE_MEMORY	1
UX_CLEAR_FEATURE	1
UX_CONFIGURATION_DESCRIPTOR_ITEM	2
UX_CONFIGURATION_DEVICE_BUS_POWERED	0x80
UX_CONFIGURATION_DEVICE_SELF_POWERED	0x40
UX_CONFIGURATION_HANDLE_UNKNOWN	0x51
UX_CONNECTION_INCOMPATIBLE	0x5a
UX_CONTROL_ENDPOINT	0
UX_CONTROL_TRANSFER_TIMEOUT	1000
UX_CONTROLLER_INIT_FAILED	0x32
UX_CONTROLLER_UNKNOWN	0x55
UX_DCD_CHANGE_STATE	19
UX_DCD_CREATE_ENDPOINT	14
UX_DCD_DESTROY_ENDPOINT	15
UX_DCD_DISABLE_CONTROLLER	1
UX_DCD_DISABLE_PORT	4
UX_DCD_ENABLE_PORT	3
UX_DCD_ENDPOINT_STATUS	21
UX_DCD_GET_FRAME_NUMBER	10
UX_DCD_GET_PORT_STATUS	2
UX_DCD_ISR_PENDING	18
UX_DCD_POWER_DOWN_PORT	6
UX_DCD_POWER_ON_PORT	5
UX_DCD_RESET_ENDPOINT	16
UX_DCD_RESET_PORT	9

UX_DCD_RESUME_PORT	8
UX_DCD_SET_DEVICE_ADDRESS	17
UX_DCD_SET_FRAME_NUMBER	11
UX_DCD_STALL_ENDPOINT	20
UX_DCD_STATUS_DEAD	2
UX_DCD_STATUS_HALTED	0
UX_DCD_STATUS_OPERATIONAL	1
UX_DCD_SUSPEND_PORT	7
UX_DCD_TRANSFER_ABORT	13
UX_DCD_TRANSFER_REQUEST	12
UX_DCD_VBUS_RESET	0
UX_DCD_VBUS_SET	1
UX_DEFAULT_HS_MPS	64
UX_DEFAULT_MPS	8
UX_DESCRIPTOR_CORRUPTED	0x42
UX_DEVICE_ADDRESS_SET_WAIT	50
UX_DEVICE_ADDRESSED	2
UX_DEVICE_ATTACHED	1
UX_DEVICE_BUS_POWERED	1
UX_DEVICE_BUS_POWERED_STATE	7
UX_DEVICE_BUS_RESET_COMPLETED	9
UX_DEVICE_CONFIGURED	3
UX_DEVICE_DESCRIPTOR_ITEM	1
UX_DEVICE_ENUMERATION_FAILURE	0x44
UX_DEVICE_HANDLE_UNKNOWN	0x50
UX_DEVICE_INSERTION	1
UX_DEVICE_QUALIFIER_DESCRIPTOR_ITEM	6
UX_DEVICE_REMOTE_WAKEUP	8
UX_DEVICE_REMOVAL	2
UX_DEVICE_RESET	0
UX_DEVICE_RESUMED	5
UX_DEVICE_SELF_POWERED	2
UX_DEVICE_SELF_POWERED_STATE	6
UX_DEVICE_SUSPENDED	4
UX_ENDPOINT_DESCRIPTOR_ITEM	5
UX_ENDPOINT_DIRECTION	0x80
UX_ENDPOINT_HALT	0
UX_ENDPOINT_HALTED	2

UX_ENDPOINT_HANDLE_UNKNOWN	0x53
UX_ENDPOINT_IN	0x80
UX_ENDPOINT_OUT	0x00
UX_ENDPOINT_RESET	0
UX_ENDPOINT_RUNNING	1
UX_ERROR	0xff
UX_FALSE	0
UX_FULL_SPEED_DEVICE	1
UX_FUNCTION_NOT_SUPPORTED	0x54
UX_GET_CONFIGURATION	8
UX_GET_DESCRIPTOR	6
UX_GET_INTERFACE	10
UX_GET_STATUS	0
UX_HCD_CREATE_ENDPOINT	14
UX_HCD_DESTROY_ENDPOINT	15
UX_HCD_DISABLE_CONTROLLER	1
UX_HCD_DISABLE_PORT	4
UX_HCD_ENABLE_PORT	3
UX_HCD_GET_FRAME_NUMBER	10
UX_HCD_GET_PORT_STATUS	2
UX_HCD_POWER_DOWN_PORT	6
UX_HCD_POWER_ON_PORT	5
UX_HCD_PROCESS_DONE_QUEUE	17
UX_HCD_RESET_ENDPOINT	16
UX_HCD_RESET_PORT	9
UX_HCD_RESUME_PORT	8
UX_HCD_SET_FRAME_NUMBER	11
UX_HCD_STATUS_DEAD	2
UX_HCD_STATUS_HALTED	0
UX_HCD_STATUS_OPERATIONAL	1
UX_HCD_SUSPEND_PORT	7
UX_HCD_TRANSFER_ABORT	13
UX_HCD_TRANSFER_REQUEST	12
UX_HID_DESCRIPTOR_ENTRIES	7
UX_HID_DESCRIPTOR_LENGTH	9
UX_HIGH_SPEED_DEVICE	2
UX_HOST_CLASS_ALREADY_INSTALLED	0x58
UX_HOST_CLASS_AUDIO_3D_STEREO_EXTENDED_PROCESS	0x03

UX_HOST_CLASS_AUDIO_AUTOMATIC_GAIN_CONTROL	0x07
UX_HOST_CLASS_AUDIO_BASS_BOOST_CONTROL	0x09
UX_HOST_CLASS_AUDIO_BASS_CONTROL	0x03
UX_HOST_CLASS_AUDIO_BIDIRECTIONAL_UNDEFINED	0x0400
UX_HOST_CLASS_AUDIO_CHORUS_PROCESS	0x05
UX_HOST_CLASS_AUDIO_CLASS	1
UX_HOST_CLASS_AUDIO_CLASS_TRANSFER_TIMEOUT	30
UX_HOST_CLASS_AUDIO_COMMUNICATION_SPEAKER	0x0306
UX_HOST_CLASS_AUDIO_COPY_PROTECT_CONTROL	0x01
UX_HOST_CLASS_AUDIO_CS_AC_UNDEFINED	0x00
UX_HOST_CLASS_AUDIO_CS_AS_GENERAL	0x01
UX_HOST_CLASS_AUDIO_CS_AS_UNDEFINED	0x00
UX_HOST_CLASS_AUDIO_CS_CONFIGURATION	0x22
UX_HOST_CLASS_AUDIO_CS_DEVICE	0x21
UX_HOST_CLASS_AUDIO_CS_ENDPOINT	0x25
UX_HOST_CLASS_AUDIO_CS_EXTENSION_UNIT	0x08
UX_HOST_CLASS_AUDIO_CS_FEATURE_UNIT	0x06
UX_HOST_CLASS_AUDIO_CS_FORMAT_SPECIFIC	0x03
UX_HOST_CLASS_AUDIO_CS_FORMAT_TYPE	0x02
UX_HOST_CLASS_AUDIO_CS_HEADER	0x01
UX_HOST_CLASS_AUDIO_CS_INPUT_TERMINAL	0x02
UX_HOST_CLASS_AUDIO_CS_INTERFACE	0x24
UX_HOST_CLASS_AUDIO_CS_MIXER_UNIT	0x04
UX_HOST_CLASS_AUDIO_CS_OUTPUT_TERMINAL	0x03
UX_HOST_CLASS_AUDIO_CS_PROCESSING_UNIT	0x07
UX_HOST_CLASS_AUDIO_CS_SELECTOR_UNIT	0x05
UX_HOST_CLASS_AUDIO_CS_STRING	0x23
UX_HOST_CLASS_AUDIO_CS_UNDEFINED	0x20
UX_HOST_CLASS_AUDIO_DELAY_CONTROL	0x08
UX_HOST_CLASS_AUDIO_DESCRIPTOR_UNDEFINED	0x00
UX_HOST_CLASS_AUDIO_DESKTOP_MICROPHONE	0x0202
UX_HOST_CLASS_AUDIO_DESKTOP_SPEAKER	0x0304
UX_HOST_CLASS_AUDIO_DOLBY_PROLOGIC_PROCESS	0x02
UX_HOST_CLASS_AUDIO_DOWN_LINE_PHONE	0x0403
UX_HOST_CLASS_AUDIO_DYN_RANGE_COMP_PROCESS	0x06
UX_HOST_CLASS_AUDIO_ECHO_CANCEL_SPEAKERPHONE	0x0405
UX_HOST_CLASS_AUDIO_ECHO_SUPRESS_SPEAKERPHONE	0x0404
UX_HOST_CLASS_AUDIO_EP_GENERAL	0x01

UX_HOST_CLASS_AUDIO_FEATURE_UNIT_DESCRIPTOR_ENTRIES	7
UX_HOST_CLASS_AUDIO_FEATURE_UNIT_DESCRIPTOR_LENGTH	7
UX_HOST_CLASS_AUDIO_FORMAT_ALAW	4
UX_HOST_CLASS_AUDIO_FORMAT_IEEE_FLOAT	3
UX_HOST_CLASS_AUDIO_FORMAT_MULAW	5
UX_HOST_CLASS_AUDIO_FORMAT_PCM	1
UX_HOST_CLASS_AUDIO_FORMAT_PCM8	2
UX_HOST_CLASS_AUDIO_FORMAT_TYPE_I	1
UX_HOST_CLASS_AUDIO_FORMAT_TYPE_II	2
UX_HOST_CLASS_AUDIO_FORMAT_TYPE_III	3
UX_HOST_CLASS_AUDIO_FORMAT_TYPE_UNDEFINED	0
UX_HOST_CLASS_AUDIO_FU_CONTROL_UNDEFINED	0x00
UX_HOST_CLASS_AUDIO_GET_CUR	0x81
UX_HOST_CLASS_AUDIO_GET_MAX	0x83
UX_HOST_CLASS_AUDIO_GET_MEM	0x85
UX_HOST_CLASS_AUDIO_GET_MIN	0x82
UX_HOST_CLASS_AUDIO_GET_RES	0x84
UX_HOST_CLASS_AUDIO_GET_STAT	0xFF
UX_HOST_CLASS_AUDIO_GRAPHIC_EQUALIZER_CONTROL	0x06
UX_HOST_CLASS_AUDIO_HANDSET	0x0401
UX_HOST_CLASS_AUDIO_HEAD_MOUNTED_DISPLAY	0x0303
UX_HOST_CLASS_AUDIO_HEADPHONES	0x0302
UX_HOST_CLASS_AUDIO_HEADSET	0x0402
UX_HOST_CLASS_AUDIO_INPUT	0x0200
UX_HOST_CLASS_AUDIO_INPUT_TERMINAL_DESCRIPTOR_ENTRIES	10
UX_HOST_CLASS_AUDIO_INPUT_TERMINAL_DESCRIPTOR_LENGTH	12
UX_HOST_CLASS_AUDIO_INTERFACE_DESCRIPTOR_ENTRIES	8
UX_HOST_CLASS_AUDIO_INTERFACE_DESCRIPTOR_LENGTH	8
UX_HOST_CLASS_AUDIO_LOUNDNESS_CONTROL	0x0A
UX_HOST_CLASS_AUDIO_LOW_FREQUENCY_SPEAKER	0x0307
UX_HOST_CLASS_AUDIO_MICROPHONE	0x0201
UX_HOST_CLASS_AUDIO_MICROPHONE_ARRAY	0x0205
UX_HOST_CLASS_AUDIO_MID_CONTROL	0x04
UX_HOST_CLASS_AUDIO_MUTE_CONTROL	0x01
UX_HOST_CLASS_AUDIO_OMNI_DIRECTIONAL_MICROPHONE	0x0204
UX_HOST_CLASS_AUDIO_OUTPUT	0x0300
UX_HOST_CLASS_AUDIO_OUTPUT_TERMINAL_DESCRIPTOR_ENTRIES	8
UX_HOST_CLASS_AUDIO_OUTPUT_TERMINAL_DESCRIPTOR_LENGTH	9

UX_HOST_CLASS_AUDIO_PERSONAL_MICROPHONE	0x0203
UX_HOST_CLASS_AUDIO_PHONE_LINE	0x0401
UX_HOST_CLASS_AUDIO_PROCESS_UNDEFINED	0x00
UX_HOST_CLASS_AUDIO_PROCESSING_MICROPHONE_ARRAY	0x0206
UX_HOST_CLASS_AUDIO_PROTOCOL_UNDEFINED	0
UX_HOST_CLASS_AUDIO_REQUEST_CODE_UNDEFINED	0x00
UX_HOST_CLASS_AUDIO_REVERBERATION_PROCESS	0x04
UX_HOST_CLASS_AUDIO_ROOM_SPEAKER	0x0305
UX_HOST_CLASS_AUDIO_SET_CUR	0x01
UX_HOST_CLASS_AUDIO_SET_MAX	0x03
UX_HOST_CLASS_AUDIO_SET_MEM	0x05
UX_HOST_CLASS_AUDIO_SET_MIN	0x02
UX_HOST_CLASS_AUDIO_SET_RES	0x04
UX_HOST_CLASS_AUDIO_SPEAKER	0x0301
UX_HOST_CLASS_AUDIO_SPEAKERPHONE	0x0403
UX_HOST_CLASS_AUDIO_STREAMING_ENDPOINT_DESCRIPTOR_ENTRIES	6
UX_HOST_CLASS_AUDIO_STREAMING_ENDPOINT_DESCRIPTOR_LENGTH	6
UX_HOST_CLASS_AUDIO_STREAMING_INTERFACE_DESCRIPTOR_ENTRIES	6
UX_HOST_CLASS_AUDIO_STREAMING_INTERFACE_DESCRIPTOR_LENGTH	6
UX_HOST_CLASS_AUDIO_SUBCLASS_CONTROL	1
UX_HOST_CLASS_AUDIO_SUBCLASS_MIDI_STREAMING	3
UX_HOST_CLASS_AUDIO_SUBCLASS_STREAMING	2
UX_HOST_CLASS_AUDIO_SUBCLASS_UNDEFINED	0
UX_HOST_CLASS_AUDIO_TE_CONTROL_UNDEFINED	0x00
UX_HOST_CLASS_AUDIO_TELEPHONE	0x0402
UX_HOST_CLASS_AUDIO_TELEPHONTY_UNDEFINED	0x0400
UX_HOST_CLASS_AUDIO_TREBLE_CONTROL	0x05
UX_HOST_CLASS_AUDIO_UP_DOWN_MIX_PROCESS	0x01
UX_HOST_CLASS_AUDIO_VOLUME_CONTROL	0x02
UX_HOST_CLASS_AUDIO_WRONG_INTERFACE	0x81
UX_HOST_CLASS_AUDIO_WRONG_TYPE	0x80
UX_HOST_CLASS_COMMAND_ACTIVATE	2
UX_HOST_CLASS_COMMAND_DEACTIVATE	3
UX_HOST_CLASS_COMMAND_QUERY	1
UX_HOST_CLASS_COMMAND_USAGE_CSP	2
UX_HOST_CLASS_COMMAND_USAGE_PIDVID	1
UX_HOST_CLASS_HID_CLASS	3
UX_HOST_CLASS_HID_COLLECTION_APPLICATION	1

UX_HOST_CLASS_HID_COLLECTION_LOGICAL	2
UX_HOST_CLASS_HID_COLLECTION_OVERFLOW	0x75
UX_HOST_CLASS_HID_COLLECTION_PHYSICAL	0
UX_HOST_CLASS_HID_COLLECTION_UNDERFLOW	0x76
UX_HOST_CLASS_HID_CONSUMER_AC_ADD_TO_CART	0x261
UX_HOST_CLASS_HID_CONSUMER_AC_ALL_CAPS	0x243
UX_HOST_CLASS_HID_CONSUMER_AC_ATTACH_COMMENT	0x26E
UX_HOST_CLASS_HID_CONSUMER_AC_ATTACH_FILE	0x28C
UX_HOST_CLASS_HID_CONSUMER_AC_BACK	0x223
UX_HOST_CLASS_HID_CONSUMER_AC_BOLD	0x23D
UX_HOST_CLASS_HID_CONSUMER_AC_BOOKMARKS	0x229
UX_HOST_CLASS_HID_CONSUMER_AC_BULLETED_LIST	0x259
UX_HOST_CLASS_HID_CONSUMER_AC_BUY_CHECKOUT	0x260
UX_HOST_CLASS_HID_CONSUMER_AC_CANCEL	0x25E
UX_HOST_CLASS_HID_CONSUMER_AC_CATALOG	0x25F
UX_HOST_CLASS_HID_CONSUMER_AC_CLEAR_ALARM	0x282
UX_HOST_CLASS_HID_CONSUMER_AC_CLOSE	0x202
UX_HOST_CLASS_HID_CONSUMER_AC_COLLAPSE	0x264
UX_HOST_CLASS_HID_CONSUMER_AC_COLLAPSE_ALL	0x265
UX_HOST_CLASS_HID_CONSUMER_AC_COPY	0x21A
UX_HOST_CLASS_HID_CONSUMER_AC_CUT	0x21B
UX_HOST_CLASS_HID_CONSUMER_AC_DELETE	0x269
UX_HOST_CLASS_HID_CONSUMER_AC_DELETE_COMMENT	0x26F
UX_HOST_CLASS_HID_CONSUMER_AC_DEMOTE	0x25B
UX_HOST_CLASS_HID_CONSUMER_AC_DISRIBUTE_HORIZONTALLY	0x29A
UX_HOST_CLASS_HID_CONSUMER_AC_DISTRIBUTE_VERTICALLY	0x29B
UX_HOST_CLASS_HID_CONSUMER_AC_DOWNLOAD	0x28E
UX_HOST_CLASS_HID_CONSUMER_AC_EDIT	0x23C
UX_HOST_CLASS_HID_CONSUMER_AC_EDIT_TIME_ZONES	0x280
UX_HOST_CLASS_HID_CONSUMER_AC_EXIT	0x203
UX_HOST_CLASS_HID_CONSUMER_AC_EXPAND	0x262
UX_HOST_CLASS_HID_CONSUMER_AC_EXPAND_ALL	0x263
UX_HOST_CLASS_HID_CONSUMER_AC_FILTER	0x27C
UX_HOST_CLASS_HID_CONSUMER_AC_FIND	0x21E
UX_HOST_CLASS_HID_CONSUMER_AC_FIND_AND_REPLACE	0x21F
UX_HOST_CLASS_HID_CONSUMER_AC_FLIP_HORIZONTAL	0x246
UX_HOST_CLASS_HID_CONSUMER_AC_FLIP_VERTICAL	0x247
UX_HOST_CLASS_HID_CONSUMER_AC_FONT_COLOR	0x24B

UX_HOST_CLASS_HID_CONSUMER_AC_FONT_SELECT	0x24A
UX_HOST_CLASS_HID_CONSUMER_AC_FONT_SIZE	0x24C
UX_HOST_CLASS_HID_CONSUMER_AC_FORMAT	0x23B
UX_HOST_CLASS_HID_CONSUMER_AC_FORWARD	0x224
UX_HOST_CLASS_HID_CONSUMER_AC_FORWARD_MSG	0x28A
UX_HOST_CLASS_HID_CONSUMER_AC_FULL_SCREEN_VIEW	0x22F
UX_HOST_CLASS_HID_CONSUMER_AC_GO_TO	0x221
UX_HOST_CLASS_HID_CONSUMER_AC_HISTORY	0x22A
UX_HOST_CLASS_HID_CONSUMER_AC_HOME	0x222
UX_HOST_CLASS_HID_CONSUMER_AC_INDENT_DECREASE	0x255
UX_HOST_CLASS_HID_CONSUMER_AC_INDENT_INCREASE	0x256
UX_HOST_CLASS_HID_CONSUMER_AC_INSERT_COLUMN	0x291
UX_HOST_CLASS_HID_CONSUMER_AC_INSERT_FILE	0x292
UX_HOST_CLASS_HID_CONSUMER_AC_INSERT_MODE	0x268
UX_HOST_CLASS_HID_CONSUMER_AC_INSERT_OBJECT	0x294
UX_HOST_CLASS_HID_CONSUMER_AC_INSERT_PICTURE	0x293
UX_HOST_CLASS_HID_CONSUMER_AC_INSERT_ROW	0x290
UX_HOST_CLASS_HID_CONSUMER_AC_INSERT_SYMBOL	0x295
UX_HOST_CLASS_HID_CONSUMER_AC_ITALICS	0x23E
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_BLOCK_H	0x250
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_BLOCK_V	0x254
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_BOTTOM	0x253
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_CENTER_H	0x24E
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_CENTER_V	0x252
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_LEFT	0x24D
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_RIGHT	0x24F
UX_HOST_CLASS_HID_CONSUMER_AC_JUSTIFY_TOP	0x251
UX_HOST_CLASS_HID_CONSUMER_AC_LOCK	0x26A
UX_HOST_CLASS_HID_CONSUMER_AC_MAXIMIZE	0x204
UX_HOST_CLASS_HID_CONSUMER_AC_MERGE	0x298
UX_HOST_CLASS_HID_CONSUMER_AC_MINIMIZE	0x205
UX_HOST_CLASS_HID_CONSUMER_AC_MIRROR_HORIZONTAL	0x248
UX_HOST_CLASS_HID_CONSUMER_AC_MIRROR_VERTICAL	0x249
UX_HOST_CLASS_HID_CONSUMER_AC_NEW	0x1B5
UX_HOST_CLASS_HID_CONSUMER_AC_NEW_WINDOW	0x238
UX_HOST_CLASS_HID_CONSUMER_AC_NEXT_LINK	0x228
UX_HOST_CLASS_HID_CONSUMER_AC_NO	0x25D
UX_HOST_CLASS_HID_CONSUMER_AC_NORMAL_VIEW	0x230

UX_HOST_CLASS_HID_CONSUMER_AC_NUMBERED_LIST	0x257
UX_HOST_CLASS_HID_CONSUMER_AC_OPEN	0x201
UX_HOST_CLASS_HID_CONSUMER_AC_PAN	0x237
UX_HOST_CLASS_HID_CONSUMER_AC_PAN_LEFT	0x235
UX_HOST_CLASS_HID_CONSUMER_AC_PAN_RIGHT	0x236
UX_HOST_CLASS_HID_CONSUMER_AC_PASTE	0x21C
UX_HOST_CLASS_HID_CONSUMER_AC_PASTE_SPECIAL	0x267
UX_HOST_CLASS_HID_CONSUMER_AC_PREVIOUS_LINK	0x227
UX_HOST_CLASS_HID_CONSUMER_AC_PRINT	0x207
UX_HOST_CLASS_HID_CONSUMER_AC_PRINT_PREVIEW	0x266
UX_HOST_CLASS_HID_CONSUMER_AC_PROMOTE	0x25A
UX_HOST_CLASS_HID_CONSUMER_AC_PROPERTIES	0x208
UX_HOST_CLASS_HID_CONSUMER_AC_PROTECT	0x26C
UX_HOST_CLASS_HID_CONSUMER_AC_REDO_REPEAT	0x278
UX_HOST_CLASS_HID_CONSUMER_AC_REFRESH	0x226
UX_HOST_CLASS_HID_CONSUMER_AC_RENAME	0x297
UX_HOST_CLASS_HID_CONSUMER_AC_REPLY	0x288
UX_HOST_CLASS_HID_CONSUMER_AC_REPLY_ALL	0x289
UX_HOST_CLASS_HID_CONSUMER_AC_RESET_ALARM	0x284
UX_HOST_CLASS_HID_CONSUMER_AC_RESIZE	0x245
UX_HOST_CLASS_HID_CONSUMER_AC_RESTART_NUMBERING	0x258
UX_HOST_CLASS_HID_CONSUMER_AC_ROTATE	0x244
UX_HOST_CLASS_HID_CONSUMER_AC_SAVE	0x206
UX_HOST_CLASS_HID_CONSUMER_AC_SAVE_CLOSE	0x296
UX_HOST_CLASS_HID_CONSUMER_AC_SCROLL	0x234
UX_HOST_CLASS_HID_CONSUMER_AC_SCROLL_DOWN	0x233
UX_HOST_CLASS_HID_CONSUMER_AC_SCROLL_UP	0x232
UX_HOST_CLASS_HID_CONSUMER_AC_SEARCH	0x220
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_ALL	0x21D
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_COLUMN	0x274
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_OBJECT	0x277
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_PARAGRAPH	0x273
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_ROW	0x275
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_SENTENCE	0x272
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_TABLE	0x276
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_TIME_ZONE	0x27F
UX_HOST_CLASS_HID_CONSUMER_AC_SELECT_WORD	0x271
UX_HOST_CLASS_HID_CONSUMER_AC_SEND	0x28B

UX_HOST_CLASS_HID_CONSUMER_AC_SEND_RECEIVE	0x286
UX_HOST_CLASS_HID_CONSUMER_AC_SEND_TO	0x287
UX_HOST_CLASS_HID_CONSUMER_AC_SET_ALARM	0x281
UX_HOST_CLASS_HID_CONSUMER_AC_SET_BORDERS	0x28F
UX_HOST_CLASS_HID_CONSUMER_AC_SET_CLOCK	0x27D
UX_HOST_CLASS_HID_CONSUMER_AC_SNOOZE_ALARM	0x283
UX_HOST_CLASS_HID_CONSUMER_AC_SORT	0x279
UX_HOST_CLASS_HID_CONSUMER_AC_SORT_ASCENDING	0x27A
UX_HOST_CLASS_HID_CONSUMER_AC_SORT_DESCENDING	0x27B
UX_HOST_CLASS_HID_CONSUMER_AC_SPLIT	0x299
UX_HOST_CLASS_HID_CONSUMER_AC_STOP	0x225
UX_HOST_CLASS_HID_CONSUMER_AC_STRIKETHROUGH	0x240
UX_HOST_CLASS_HID_CONSUMER_AC_SUBSCRIPT	0x241
UX_HOST_CLASS_HID_CONSUMER_AC_SUBSCRIPTIONS	0x22B
UX_HOST_CLASS_HID_CONSUMER_AC_SUPERSCRIPT	0x242
UX_HOST_CLASS_HID_CONSUMER_AC_SYNCHRONIZE	0x285
UX_HOST_CLASS_HID_CONSUMER_AC_TILE_HORIZONTALLY	0x239
UX_HOST_CLASS_HID_CONSUMER_AC_TILE_VERTICALLY	0x23A
UX_HOST_CLASS_HID_CONSUMER_AC_UNDERLINE	0x23F
UX_HOST_CLASS_HID_CONSUMER_AC_UNDO	0x209
UX_HOST_CLASS_HID_CONSUMER_AC_UNLOCK	0x26B
UX_HOST_CLASS_HID_CONSUMER_AC_UNPROTECT	0x26D
UX_HOST_CLASS_HID_CONSUMER_AC_UPLOAD	0x28D
UX_HOST_CLASS_HID_CONSUMER_AC_VIEW_CLOCK	0x27E
UX_HOST_CLASS_HID_CONSUMER_AC_VIEW_COMMENT	0x270
UX_HOST_CLASS_HID_CONSUMER_AC_VIEW_TOGGLE	0x231
UX_HOST_CLASS_HID_CONSUMER_AC_YES	0x25C
UX_HOST_CLASS_HID_CONSUMER_AC_ZOOM	0x22E
UX_HOST_CLASS_HID_CONSUMER_AC_ZOOM_IN	0x22C
UX_HOST_CLASS_HID_CONSUMER_AC_ZOOM_OUT	0x22D
UX_HOST_CLASS_HID_CONSUMER_AL_A_V_CAPTURE_PLAYBACK	0x192
UX_HOST_CLASS_HID_CONSUMER_AL_ALARMS	0x1B1
UX_HOST_CLASS_HID_CONSUMER_AL_CALCULATOR	0x191
UX_HOST_CLASS_HID_CONSUMER_AL_CALENDAR_SCHEDULE	0x18D
UX_HOST_CLASS_HID_CONSUMER_AL_CHECKBOOK_FINANCE	0x190
UX_HOST_CLASS_HID_CONSUMER_AL_CLOCK	0x1B2
UX_HOST_CLASS_HID_CONSUMER_AL_COMMAND_LINE_PROCESSOR	0x19F

UX_HOST_CLASS_HID_CONSUMER_AL_CONSUMER_CONTROL_CONFIGURATION	0x182
UX_HOST_CLASS_HID_CONSUMER_AL_CONTACTS_ADDRESS_BOOK	0x18C
UX_HOST_CLASS_HID_CONSUMER_AL_CONTROL_PANEL	0x19E
UX_HOST_CLASS_HID_CONSUMER_AL_DATABASE_APP	0x188
UX_HOST_CLASS_HID_CONSUMER_AL_DESKTOP	0x1A9
UX_HOST_CLASS_HID_CONSUMER_AL_DICTIONARY	0x1A8
UX_HOST_CLASS_HID_CONSUMER_AL_DOCUMENTS	0x1A6
UX_HOST_CLASS_HID_CONSUMER_AL_EMAIL_READER	0x189
UX_HOST_CLASS_HID_CONSUMER_AL_ENCRYPTION	0x1AF
UX_HOST_CLASS_HID_CONSUMER_AL_FILE_BROWSER	0x1B3
UX_HOST_CLASS_HID_CONSUMER_AL_GRAMMAR_CHECK	0x1AB
UX_HOST_CLASS_HID_CONSUMER_AL_GRAPHICS_EDITOR	0x186
UX_HOST_CLASS_HID_CONSUMER_AL_INTEGRATED_HELP_CENTER	0x1A5
UX_HOST_CLASS_HID_CONSUMER_AL_INTERNET_BROWSER	0x195
UX_HOST_CLASS_HID_CONSUMER_AL_KEYBOARD_LAYOUT	0x1AD
UX_HOST_CLASS_HID_CONSUMER_AL_LAN_WAN_BROWSER	0x194
UX_HOST_CLASS_HID_CONSUMER_AL_LAUNCH_BUTTON_CONFIGURATION	0x180
UX_HOST_CLASS_HID_CONSUMER_AL_LOCAL_MACHINE_BROWSER	0x193
UX_HOST_CLASS_HID_CONSUMER_AL_LOG_JOURNAL_TIMECARD	0x18F
UX_HOST_CLASS_HID_CONSUMER_AL_LOGOFF	0x19B
UX_HOST_CLASS_HID_CONSUMER_AL_LOGON	0x19A
UX_HOST_CLASS_HID_CONSUMER_AL_LOGON_LOGOFF	0x19C
UX_HOST_CLASS_HID_CONSUMER_AL_NETWORK_CHAT	0x198
UX_HOST_CLASS_HID_CONSUMER_AL_NETWORK_CONFERENCE	0x197
UX_HOST_CLASS_HID_CONSUMER_AL_NEWSREADER	0x18A
UX_HOST_CLASS_HID_CONSUMER_AL_NEXT_APPLICATION	0x1A2
UX_HOST_CLASS_HID_CONSUMER_AL_POWER_STATUS	0x1B4
UX_HOST_CLASS_HID_CONSUMER_AL_PREEMPTIVE_HALT_APPLICATION	0x1A4
UX_HOST_CLASS_HID_CONSUMER_AL_PRESENTATION_APP	0x187
UX_HOST_CLASS_HID_CONSUMER_AL_PREVIOUS_APPLICATION	0x1A3
UX_HOST_CLASS_HID_CONSUMER_AL_PROCESS_MANAGER	0x1A0
UX_HOST_CLASS_HID_CONSUMER_AL_PROGRAMMABLE_BUTTON	0x181
UX_HOST_CLASS_HID_CONSUMER_AL_REMOTE_NETWORKING	0x196
UX_HOST_CLASS_HID_CONSUMER_AL_SCREEN_SAVER	0x1B0
UX_HOST_CLASS_HID_CONSUMER_AL_SCREENSAVER	0x19D
UX_HOST_CLASS_HID_CONSUMER_AL_SELECT_APPLICATION	0x1A1

UX_HOST_CLASS_HID_CONSUMER_AL_SPELL_CHECK	0x1AA
UX_HOST_CLASS_HID_CONSUMER_AL_SPREADSHEET	0x185
UX_HOST_CLASS_HID_CONSUMER_AL_TASK_PROJECT_MANAGER	0x18E
UX_HOST_CLASS_HID_CONSUMER_AL_TELEPHONY_DIALER	0x199
UX_HOST_CLASS_HID_CONSUMER_AL_TEXT_EDITOR	0x184
UX_HOST_CLASS_HID_CONSUMER_AL_THESAURUS	0x1A7
UX_HOST_CLASS_HID_CONSUMER_AL_VIRUS_PROTECTION	0x1AE
UX_HOST_CLASS_HID_CONSUMER_AL_VOICEMAIL	0x18B
UX_HOST_CLASS_HID_CONSUMER_AL_WIRELESS_STATUS	0x1AC
UX_HOST_CLASS_HID_CONSUMER_AL_WORD_PROCESSOR	0x183
UX_HOST_CLASS_HID_CONSUMER_ALTERNATE_AUDIO_DECREMENT	0x173
UX_HOST_CLASS_HID_CONSUMER_ALTERNATE_AUDIO_INCREMENT	0x172
UX_HOST_CLASS_HID_CONSUMER_AM_PM	0x22
UX_HOST_CLASS_HID_CONSUMER_APPLICATION_LAUNCH_BUTTONS	0x174
UX_HOST_CLASS_HID_CONSUMER_ASSIGN_SELECTION	0x81
UX_HOST_CLASS_HID_CONSUMER_BALANCE	0xE1
UX_HOST_CLASS_HID_CONSUMER_BALANCE_LEFT	0x150
UX_HOST_CLASS_HID_CONSUMER_BALANCE_RIGHT	0x10D
UX_HOST_CLASS_HID_CONSUMER_BASS	0xE3
UX_HOST_CLASS_HID_CONSUMER_BASS_BOOST	0xE5
UX_HOST_CLASS_HID_CONSUMER_BASS_DECREMENT	0x152
UX_HOST_CLASS_HID_CONSUMER_BASS_INCREMENT	0x151
UX_HOST_CLASS_HID_CONSUMER_BROADCAST_MODE	0x64
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_CENTER	0x162
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_CENTER_FRONT	0x164
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_DECREMENT	0x9D
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_FRONT	0x163
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_INCREMENT	0x9C
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_L	0x86
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_LEFT	0x160
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_LOW_FREQUENCY	0x167
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_RIGHT	0x161
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_SIDE	0x165
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_SURROUND	0x166
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_TOP	0x168
UX_HOST_CLASS_HID_CONSUMER_CHANNEL_UNKNOWN	0x169
UX_HOST_CLASS_HID_CONSUMER_CLEAR_MARK	0xC3
UX_HOST_CLASS_HID_CONSUMER_CLIMATE_CONTROL_ENABLE	0x103

UX_HOST_CLASS_HID_CONSUMER_CLOSED_CAPTION	0x61
UX_HOST_CLASS_HID_CONSUMER_CLOSED_CAPTION_SELECT	0x62
UX_HOST_CLASS_HID_CONSUMER_COUNTER_RESET	0xC8
UX_HOST_CLASS_HID_CONSUMER_DAILY	0xA2
UX_HOST_CLASS_HID_CONSUMER_DATA_ON_SCREEN	0x60
UX_HOST_CLASS_HID_CONSUMER_DURESS_ALARM	0x10A
UX_HOST_CLASS_HID_CONSUMER_EJECT	0xB8
UX_HOST_CLASS_HID_CONSUMER_ENTER_CHANNEL	0x84
UX_HOST_CLASS_HID_CONSUMER_ENTER_DISC	0xBB
UX_HOST_CLASS_HID_CONSUMER_EXTENDED_PLAY	0xF4
UX_HOST_CLASS_HID_CONSUMER_FAN_ENABLE	0xF6
UX_HOST_CLASS_HID_CONSUMER_FAN_SPEED	0x100
UX_HOST_CLASS_HID_CONSUMER_FAST_FORWARD	0xB3
UX_HOST_CLASS_HID_CONSUMER_FIRE_ALARM	0x106
UX_HOST_CLASS_HID_CONSUMER_FRAME_BACK	0xC1
UX_HOST_CLASS_HID_CONSUMER_FRAME_FORWARD	0xC0
UX_HOST_CLASS_HID_CONSUMER_FUNCTION_BUTTONS	0x36
UX_HOST_CLASS_HID_CONSUMER_GRAPHIC_EQUALIZER	0x06
UX_HOST_CLASS_HID_CONSUMER_HEADPHONE	0x05
UX_HOST_CLASS_HID_CONSUMER_HELP	0x95
UX_HOST_CLASS_HID_CONSUMER_HOLDUP_ALARM	0x10B
UX_HOST_CLASS_HID_CONSUMER_ILLUMINATION	0x35
UX_HOST_CLASS_HID_CONSUMER_LIGHT_ENABLE	0x101
UX_HOST_CLASS_HID_CONSUMER_LIGHT_ILLUMINATION_LEVEL	0x102
UX_HOST_CLASS_HID_CONSUMER_LONG_PLAY	0xF3
UX_HOST_CLASS_HID_CONSUMER_LOUDNESS	0xE7
UX_HOST_CLASS_HID_CONSUMER_MARK	0xC2
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT	0x9E
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_CABLE	0x97
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_CALL	0x9B
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_CD	0x91
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_COMPUTER	0x88
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_DVD	0x8B
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_GAMES	0x8F
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_HOME	0x9A
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_MESSAGES	0x90
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_PROGRAM_GUIDE	0x8D
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_SATELLITE	0x98

UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_SECURITY	0x99
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_TAPE	0x96
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_TELEPHONE	0x8C
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_TUNER	0x93
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_TV	0x89
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_VCR	0x92
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_VIDEO_PHONE	0x8E
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECT_WWW	0x8A
UX_HOST_CLASS_HID_CONSUMER_MEDIA_SELECTION	0x87
UX_HOST_CLASS_HID_CONSUMER_MEDICAL_ALARM	0x10C
UX_HOST_CLASS_HID_CONSUMER_MENU	0x40
UX_HOST_CLASS_HID_CONSUMER_MENU_DOWN	0x43
UX_HOST_CLASS_HID_CONSUMER_MENU_ESCAPE	0x46
UX_HOST_CLASS_HID_CONSUMER_MENU_LEFT	0x44
UX_HOST_CLASS_HID_CONSUMER_MENU_PICK	0x41
UX_HOST_CLASS_HID_CONSUMER_MENU_RIGHT	0x45
UX_HOST_CLASS_HID_CONSUMER_MENU_UP	0x42
UX_HOST_CLASS_HID_CONSUMER_MENU_VALUE_DECREASE	0x48
UX_HOST_CLASS_HID_CONSUMER_MENU_VALUE_INCREASE	0x47
UX_HOST_CLASS_HID_CONSUMER_MICROPHONE	0x04
UX_HOST_CLASS_HID_CONSUMER_MODE_STEP	0x82
UX_HOST_CLASS_HID_CONSUMER_MONTHLY	0xA4
UX_HOST_CLASS_HID_CONSUMER_MOTION	0x109
UX_HOST_CLASS_HID_CONSUMER_MPX	0xE8
UX_HOST_CLASS_HID_CONSUMER_MUTE	0xE2
UX_HOST_CLASS_HID_CONSUMER_NUMERIC_KEY_PAD	0x02
UX_HOST_CLASS_HID_CONSUMER_ONCE	0xA1
UX_HOST_CLASS_HID_CONSUMER_ORDER_MOVIE	0x85
UX_HOST_CLASS_HID_CONSUMER_PAUSE	0xB1
UX_HOST_CLASS_HID_CONSUMER_PLAY	0xB0
UX_HOST_CLASS_HID_CONSUMER_PLAY_PAUSE	0xCD
UX_HOST_CLASS_HID_CONSUMER_PLAY_SKIP	0xCE
UX_HOST_CLASS_HID_CONSUMER_PLAYBACK_SPEED	0xF1
UX_HOST_CLASS_HID_CONSUMER_PLUS_10	0x20
UX_HOST_CLASS_HID_CONSUMER_PLUS_100	0x21
UX_HOST_CLASS_HID_CONSUMER_POLICE_ALARM	0x107
UX_HOST_CLASS_HID_CONSUMER_POWER	0x30
UX_HOST_CLASS_HID_CONSUMER_PROGRAMMABLE_BUTTONS	0x03

UX_HOST_CLASS_HID_CONSUMER_PROXIMITY	0x108
UX_HOST_CLASS_HID_CONSUMER_QUIT	0x94
UX_HOST_CLASS_HID_CONSUMER_RANDOM_PLAY	0xB9
UX_HOST_CLASS_HID_CONSUMER_RECALL_LAST	0x83
UX_HOST_CLASS_HID_CONSUMER_RECORD	0xB2
UX_HOST_CLASS_HID_CONSUMER_REMOTE_CONTROL	0x01
UX_HOST_CLASS_HID_CONSUMER_REPEAT	0xBC
UX_HOST_CLASS_HID_CONSUMER_REPEAT_FROM_MARK	0xC4
UX_HOST_CLASS_HID_CONSUMER_RESET	0x31
UX_HOST_CLASS_HID_CONSUMER_RETURN_TO_MARK	0xC5
UX_HOST_CLASS_HID_CONSUMER_REWIND	0xB4
UX_HOST_CLASS_HID_CONSUMER_ROOM_TEMPERATURE	0x104
UX_HOST_CLASS_HID_CONSUMER_SCAN_NEXT_TRACK	0xB5
UX_HOST_CLASS_HID_CONSUMER_SCAN_PREVIOUS_TRACK	0xB6
UX_HOST_CLASS_HID_CONSUMER_SEARCH_MARK_BACKWARDS	0xC7
UX_HOST_CLASS_HID_CONSUMER_SEARCH_MARK_FORWARD	0xC6
UX_HOST_CLASS_HID_CONSUMER_SECURITY_ENABLE	0x105
UX_HOST_CLASS_HID_CONSUMER_SELECT_DISC	0xBA
UX_HOST_CLASS_HID_CONSUMER_SELECTION	0x80
UX_HOST_CLASS_HID_CONSUMER_SHOW_COUNTER	0xC9
UX_HOST_CLASS_HID_CONSUMER_SLEEP	0x32
UX_HOST_CLASS_HID_CONSUMER_SLEEP_AFTER	0x33
UX_HOST_CLASS_HID_CONSUMER_SLEEP_MODE_RTC	0x34
UX_HOST_CLASS_HID_CONSUMER_SLOW	0xF5
UX_HOST_CLASS_HID_CONSUMER_SLOW_TRACKING	0xBF
UX_HOST_CLASS_HID_CONSUMER_SNAPSHOT	0x65
UX_HOST_CLASS_HID_CONSUMER_SPEAKER_SYSTEM	0x155
UX_HOST_CLASS_HID_CONSUMER_SPEED_SELECT	0xF0
UX_HOST_CLASS_HID_CONSUMER_STANDARD_PLAY	0xF2
UX_HOST_CLASS_HID_CONSUMER_STILL	0x66
UX_HOST_CLASS_HID_CONSUMER_STOP	0xB7
UX_HOST_CLASS_HID_CONSUMER_STOP_EJECT	0xCC
UX_HOST_CLASS_HID_CONSUMER_SUB_CHANNEL	0x16A
UX_HOST_CLASS_HID_CONSUMER_SUB_CHANNEL_DECREMENT	0x171
UX_HOST_CLASS_HID_CONSUMER_SUB_CHANNEL_INCREMENT	0x170
UX_HOST_CLASS_HID_CONSUMER_SURROUND_MODE	0xE6
UX_HOST_CLASS_HID_CONSUMER_TRACK_NORMAL	0xBE
UX_HOST_CLASS_HID_CONSUMER_TRACKING	0xBD

UX_HOST_CLASS_HID_CONSUMER_TRACKING_DECREMENT	0xCB
UX_HOST_CLASS_HID_CONSUMER_TRACKING_INCREMENT	0xCA
UX_HOST_CLASS_HID_CONSUMER_TREBLE	0xE4
UX_HOST_CLASS_HID_CONSUMER_TREBLE_DECREMENT	0x154
UX_HOST_CLASS_HID_CONSUMER_TREBLE_INCREMENT	0x153
UX_HOST_CLASS_HID_CONSUMER_UNASSIGNED	0x00
UX_HOST_CLASS_HID_CONSUMER_VCR_PLUS	0xA0
UX_HOST_CLASS_HID_CONSUMER_VCR_TV	0x63
UX_HOST_CLASS_HID_CONSUMER_VOLUME	0xE0
UX_HOST_CLASS_HID_CONSUMER_VOLUME_DECREMENT	0xEA
UX_HOST_CLASS_HID_CONSUMER_VOLUME_INCREMENT	0xE9
UX_HOST_CLASS_HID_CONSUMER_WEEKLY	0xA3
UX_HOST_CLASS_HID_DECOMPRESSION_BUFFER	4096
UX_HOST_CLASS_HID_DELIMITER_CLOSE	0
UX_HOST_CLASS_HID_DELIMITER_ERROR	0x78
UX_HOST_CLASS_HID_DELIMITER_OPEN	1
UX_HOST_CLASS_HID_DESCRIPTOR	0x21
UX_HOST_CLASS_HID_DESCRIPTOR	0x21
UX_HOST_CLASS_HID_FIELDS	16
UX_HOST_CLASS_HID_GAME_CONTROL_3D_GAME_CONTROLLER	0x01
UX_HOST_CLASS_HID_GAME_CONTROL_BUMP	0x2C
UX_HOST_CLASS_HID_GAME_CONTROL_FLIPPER	0x2A
UX_HOST_CLASS_HID_GAME_CONTROL_GAMEAD_FIRE_JUMP	0x37
UX_HOST_CLASS_HID_GAME_CONTROL_GAMEPAD_TRIGGER	0x39
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_AUTOMATIC	0x35
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_BOLT	0x30
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_BURST	0x34
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_CLIP	0x31
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_DEVICE	0x03
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_SAFETY	0x36
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_SELECTOR	0x32
UX_HOST_CLASS_HID_GAME_CONTROL_GUN_SINGLE_SHOT	0x33
UX_HOST_CLASS_HID_GAME_CONTROL_HEIGHT_OF_POV	0x29
UX_HOST_CLASS_HID_GAME_CONTROL_LEAN_FORWARD_BACKWARD	0x28
UX_HOST_CLASS_HID_GAME_CONTROL_LEAN_RIGHT_LEFT	0x27
UX_HOST_CLASS_HID_GAME_CONTROL_MOVE_FORWARD_BACKWARD	0x25
UX_HOST_CLASS_HID_GAME_CONTROL_MOVE_RIGHT_LEFT	0x24
UX_HOST_CLASS_HID_GAME_CONTROL_MOVE_UP_DOWN	0x26

UX_HOST_CLASS_HID_GAME_CONTROL_NEW_GAME	0x2D
UX_HOST_CLASS_HID_GAME_CONTROL_PINBALL_DEVICE	0x02
UX_HOST_CLASS_HID_GAME_CONTROL_PITCH_FORWARD_BACKWARD	0x22
UX_HOST_CLASS_HID_GAME_CONTROL_PLAYER	0x2F
UX_HOST_CLASS_HID_GAME_CONTROL_POINT_OF_VIEW	0x20
UX_HOST_CLASS_HID_GAME_CONTROL_ROLL_RIGHT_LEFT	0x23
UX_HOST_CLASS_HID_GAME_CONTROL_SECONDARY_FLIPPER	0x2B
UX_HOST_CLASS_HID_GAME_CONTROL_SHOOT_BALL	0x2E
UX_HOST_CLASS_HID_GAME_CONTROL_TURN_RIGHT_LEFT	0x21
UX_HOST_CLASS_HID_GAME_CONTROL_UNDEFINED	0x00
UX_HOST_CLASS_HID_GENERIC_DESKTOP_APPLICAION_BREAK	0xA5
UX_HOST_CLASS_HID_GENERIC_DESKTOP_APPLICATION_DEBUGGER_BREAK	0xA6
UX_HOST_CLASS_HID_GENERIC_DESKTOP_BYTE_COUNT	0x3B
UX_HOST_CLASS_HID_GENERIC_DESKTOP_COUNTED_BUFFER	0x3A
UX_HOST_CLASS_HID_GENERIC_DESKTOP_D_PAD_DOWN	0x91
UX_HOST_CLASS_HID_GENERIC_DESKTOP_D_PAD_LEFT	0x93
UX_HOST_CLASS_HID_GENERIC_DESKTOP_D_PAD_RIGHT	0x92
UX_HOST_CLASS_HID_GENERIC_DESKTOP_D_PAD_UP	0x90
UX_HOST_CLASS_HID_GENERIC_DESKTOP_DIAL	0x37
UX_HOST_CLASS_HID_GENERIC_DESKTOP_FEATURE_NOTIFICATION	0x47
UX_HOST_CLASS_HID_GENERIC_DESKTOP_GAME PAD	0x05
UX_HOST_CLASS_HID_GENERIC_DESKTOP_HAT_SWITCH	0x39
UX_HOST_CLASS_HID_GENERIC_DESKTOP_JOYSTICK	0x04
UX_HOST_CLASS_HID_GENERIC_DESKTOP_KEYBOARD	0x06
UX_HOST_CLASS_HID_GENERIC_DESKTOP_KEYPAD	0x07
UX_HOST_CLASS_HID_GENERIC_DESKTOP_MOTION_WAKEUP	0x3C
UX_HOST_CLASS_HID_GENERIC_DESKTOP_MOUSE	0x02
UX_HOST_CLASS_HID_GENERIC_DESKTOP_MULTI_AXIS_CONTROLLER	0x08
UX_HOST_CLASS_HID_GENERIC_DESKTOP_POINTER	0x01
UX_HOST_CLASS_HID_GENERIC_DESKTOP_RESERVED	0x03
UX_HOST_CLASS_HID_GENERIC_DESKTOP_RX	0x33
UX_HOST_CLASS_HID_GENERIC_DESKTOP_RY	0x34
UX_HOST_CLASS_HID_GENERIC_DESKTOP_RZ	0x35
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SELECT	0x3E
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SLIDER	0x36
UX_HOST_CLASS_HID_GENERIC_DESKTOP_START	0x3D
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_APP_MENU	0x86

UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_BREAK	0xA3
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_COLD_RESTART	0x8E
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_CONTEXT_MENU	0x84
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_CONTROL	0x80
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DEBUGGER_BREAK	0xA4
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_BOTH	0xB3
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_DUAL	0xB4
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_EXTERNAL	0xB2
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_INTERNAL	0xB1
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_INVERT	0xB0
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_LCD_AUTOSCALE	0xB7
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_SWAP	0xB6
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DISPLAY_TOGGLE	0xB5
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_DOCK	0xA0
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_HIBERNATE	0xA8
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MAIN_MENU	0x85
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MENU_DOWN	0x8D
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MENU_EXIT	0x88
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MENU_HELP	0x87
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MENU_LEFT	0x8B
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MENU_RIGHT	0x8A
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MENU_SELECT	0x89
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_MENU_UP	0x8C
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_POWER_DOWN	0x81
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_SETUP	0xA2
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_SLEEP	0x82
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_SPEAKER_MUTE	0xA7
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_UNDOCK	0xA1
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_WAKE_UP	0x83
UX_HOST_CLASS_HID_GENERIC_DESKTOP_SYSTEM_WARM_RESTART	0x8F
UX_HOST_CLASS_HID_GENERIC_DESKTOP_UNDEFINED	0x00
UX_HOST_CLASS_HID_GENERIC_DESKTOP_VBRX	0x43
UX_HOST_CLASS_HID_GENERIC_DESKTOP_VBRY	0x44
UX_HOST_CLASS_HID_GENERIC_DESKTOP_VBRZ	0x45
UX_HOST_CLASS_HID_GENERIC_DESKTOP_VNO	0x46
UX_HOST_CLASS_HID_GENERIC_DESKTOP_VX	0x40
UX_HOST_CLASS_HID_GENERIC_DESKTOP_VY	0x41

UX_HOST_CLASS_HID_GENERIC_DESKTOP_VZ	0x42
UX_HOST_CLASS_HID_GENERIC_DESKTOP_WHEEL	0x38
UX_HOST_CLASS_HID_GENERIC_DESKTOP_X	0x30
UX_HOST_CLASS_HID_GENERIC_DESKTOP_Y	0x31
UX_HOST_CLASS_HID_GENERIC_DESKTOP_Z	0x32
UX_HOST_CLASS_HID_GET_IDLE	0x02
UX_HOST_CLASS_HID_GET_PROTOCOL	0x03
UX_HOST_CLASS_HID_GET_REPORT	0x01
UX_HOST_CLASS_HID_GLOBAL_TAG_LOGICAL_MAXIMUM	0x2
UX_HOST_CLASS_HID_GLOBAL_TAG_LOGICAL_MINIMUM	0x1
UX_HOST_CLASS_HID_GLOBAL_TAG_PHYSICAL_MAXIMUM	0x4
UX_HOST_CLASS_HID_GLOBAL_TAG_PHYSICAL_MINIMUM	0x3
UX_HOST_CLASS_HID_GLOBAL_TAG_POP	0xb
UX_HOST_CLASS_HID_GLOBAL_TAG_PUSH	0xa
UX_HOST_CLASS_HID_GLOBAL_TAG_REPORT_COUNT	0x9
UX_HOST_CLASS_HID_GLOBAL_TAG_REPORT_ID	0x8
UX_HOST_CLASS_HID_GLOBAL_TAG_REPORT_SIZE	0x7
UX_HOST_CLASS_HID_GLOBAL_TAG_UNIT	0x6
UX_HOST_CLASS_HID_GLOBAL_TAG_UNIT_EXPONENT	0x5
UX_HOST_CLASS_HID_GLOBAL_TAG_USAGE_PAGE	0x0
UX_HOST_CLASS_HID_INTERRUPT_ENDPOINT_ACTIVE	2
UX_HOST_CLASS_HID_INTERRUPT_ENDPOINT_READY	1
UX_HOST_CLASS_HID_ITEM_BUFFERED_BYTES	0x0100
UX_HOST_CLASS_HID_ITEM_CONSTANT	0x0001
UX_HOST_CLASS_HID_ITEM_LENGTH_MASK	3
UX_HOST_CLASS_HID_ITEM_NO_PREFERRED_STATE	0x0020
UX_HOST_CLASS_HID_ITEM_NON_LINEAR	0x0010
UX_HOST_CLASS_HID_ITEM_NULL_STATE	0x0040
UX_HOST_CLASS_HID_ITEM_RELATIVE	0x0004
UX_HOST_CLASS_HID_ITEM_TAG_LONG	3
UX_HOST_CLASS_HID_ITEM_TAG_MASK	0xf0
UX_HOST_CLASS_HID_ITEM_TAG_SHORT	1
UX_HOST_CLASS_HID_ITEM_VARIABLE	0x0002
UX_HOST_CLASS_HID_ITEM_VOLATILE	0x0080
UX_HOST_CLASS_HID_ITEM_WRAP	0x0008
UX_HOST_CLASS_HID_LED_BATTERY_LOW	0x1D
UX_HOST_CLASS_HID_LED_BATTERY_OK	0x1C
UX_HOST_CLASS_HID_LED_BATTERY_OPERATION	0x1B

UX_HOST_CLASS_HID_LED_BUSY	0x2C
UX_HOST_CLASS_HID_LED_CALL_PICKUP	0x25
UX_HOST_CLASS_HID_LED_CAMERA_OFF	0x29
UX_HOST_CLASS_HID_LED_CAMERA_ON	0x28
UX_HOST_CLASS_HID_LED_CAPS_LOCK	0x02
UX_HOST_CLASS_HID_LED_CAV	0x14
UX_HOST_CLASS_HID_LED_CLV	0x15
UX_HOST_CLASS_HID_LED_COMPOSE	0x04
UX_HOST_CLASS_HID_LED_CONFERENCE	0x26
UX_HOST_CLASS_HID_LED_COVERAGE	0x22
UX_HOST_CLASS_HID_LED_DATA_MODE	0x1A
UX_HOST_CLASS_HID_LED_DO_NOT_DISTURB	0x08
UX_HOST_CLASS_HID_LED_EQUALIZER_ENABLE	0x0D
UX_HOST_CLASS_HID_LED_ERROR	0x39
UX_HOST_CLASS_HID_LED_EXTERNAL_POWER_CONNECTED	0x4D
UX_HOST_CLASS_HID_LED_FAST_BLINK_OFF_TIME	0x46
UX_HOST_CLASS_HID_LED_FAST_BLINK_ON_TIME	0x45
UX_HOST_CLASS_HID_LED_FAST_FORWARD	0x35
UX_HOST_CLASS_HID_LED_FLASH_ON_TIME	0x42
UX_HOST_CLASS_HID_LED_FORWARD	0x31
UX_HOST_CLASS_HID_LED_GENERIC_INDICATOR	0x4B
UX_HOST_CLASS_HID_LED_HEAD_SET	0x1F
UX_HOST_CLASS_HID_LED_HIGH_CUT_FILTER	0x0B
UX_HOST_CLASS_HID_LED_HOLD	0x20
UX_HOST_CLASS_HID_LED_INDICATOR_AMBER	0x4A
UX_HOST_CLASS_HID_LED_INDICATOR_FAST_BLINK	0x40
UX_HOST_CLASS_HID_LED_INDICATOR_FLASH	0x3E
UX_HOST_CLASS_HID_LED_INDICATOR_GREEN	0x49
UX_HOST_CLASS_HID_LED_INDICATOR_OFF	0x41
UX_HOST_CLASS_HID_LED_INDICATOR_ON	0x3D
UX_HOST_CLASS_HID_LED_INDICATOR_RED	0x48
UX_HOST_CLASS_HID_LED_INDICATOR_SLOW_BLINK	0x3F
UX_HOST_CLASS_HID_LED_KANA	0x05
UX_HOST_CLASS_HID_LED_LOW_CUT_FILTER	0x0C
UX_HOST_CLASS_HID_LED_MESSAGE_WAITING	0x19
UX_HOST_CLASS_HID_LED_MICROPHONE	0x21
UX_HOST_CLASS_HID_LED_MUTE	0x09
UX_HOST_CLASS_HID_LED_NIGHT_MODE	0x23

UX_HOST_CLASS_HID_LED_NUM_LOCK	0x01
UX_HOST_CLASS_HID_LED_OFF_HOOK	0x17
UX_HOST_CLASS_HID_LED_OFF_LINE	0x2B
UX_HOST_CLASS_HID_LED_ON_LINE	0x2A
UX_HOST_CLASS_HID_LED_PAPER_JAM	0x2F
UX_HOST_CLASS_HID_LED_PAPER_OUT	0x2E
UX_HOST_CLASS_HID_LED_PAUSE	0x37
UX_HOST_CLASS_HID_LED_PLAY	0x36
UX_HOST_CLASS_HID_LED_POWER	0x06
UX_HOST_CLASS_HID_LED_READY	0x2D
UX_HOST_CLASS_HID_LED_RECORD	0x38
UX_HOST_CLASS_HID_LED_RECORDING_FORMAT_DETECT	0x16
UX_HOST_CLASS_HID_LED_REMOTE	0x30
UX_HOST_CLASS_HID_LED_REPEAT	0x10
UX_HOST_CLASS_HID_LED_REVERSE	0x32
UX_HOST_CLASS_HID_LED_REWIND	0x34
UX_HOST_CLASS_HID_LED_RING	0x18
UX_HOST_CLASS_HID_LED_SAMPLING_RATE_DETECT	0x12
UX_HOST_CLASS_HID_LED_SCROLL_LOCK	0x03
UX_HOST_CLASS_HID_LED_SEND_CALLS	0x24
UX_HOST_CLASS_HID_LED_SHIFT	0x07
UX_HOST_CLASS_HID_LED_SLOW_BLINK_OFF_TIME	0x44
UX_HOST_CLASS_HID_LED_SLOW_BLINK_ON_TIME	0x43
UX_HOST_CLASS_HID_LED_SOUND_FIELD_ON	0x0E
UX_HOST_CLASS_HID_LED_SPEAKER	0x1E
UX_HOST_CLASS_HID_LED_SPINNING	0x13
UX_HOST_CLASS_HID_LED_STAND_BY	0x27
UX_HOST_CLASS_HID_LED_STEREO	0x11
UX_HOST_CLASS_HID_LED_STOP	0x33
UX_HOST_CLASS_HID_LED_SURROUND_ON	0x0F
UX_HOST_CLASS_HID_LED_SYSTEM_SUSPEND	0x4C
UX_HOST_CLASS_HID_LED_TONE_ENABLE	0x0A
UX_HOST_CLASS_HID_LED_UNDEFINED	0x00
UX_HOST_CLASS_HID_LED_USAGE MULTI_MODE_INDICATOR	0x3C
UX_HOST_CLASS_HID_LED_USAGE_IN_USE_INDICATOR	0x3B
UX_HOST_CLASS_HID_LED_USAGE_INDICATOR_COLOR	0x47
UX_HOST_CLASS_HID_LED_USAGE_SELECTED_INDICATOR	0x3A
UX_HOST_CLASS_HID_LOCAL_TAG_DELIMITER	0xa

UX_HOST_CLASS_HID_LOCAL_TAG_DESIGNATOR_INDEX	0x3
UX_HOST_CLASS_HID_LOCAL_TAG_DESIGNATOR_MAXIMUM	0x5
UX_HOST_CLASS_HID_LOCAL_TAG_DESIGNATOR_MINIMUM	0x4
UX_HOST_CLASS_HID_LOCAL_TAG_STRING_INDEX	0x7
UX_HOST_CLASS_HID_LOCAL_TAG_STRING_MAXIMUM	0x9
UX_HOST_CLASS_HID_LOCAL_TAG_STRING_MINIMUM	0x8
UX_HOST_CLASS_HID_LOCAL_TAG_USAGE	0x0
UX_HOST_CLASS_HID_LOCAL_TAG_USAGE_MAXIMUM	0x2
UX_HOST_CLASS_HID_LOCAL_TAG_USAGE_MINIMUM	0x1
UX_HOST_CLASS_HID_MAIN_TAG_COLLECTION	0xa
UX_HOST_CLASS_HID_MAIN_TAG_END_COLLECTION	0xc
UX_HOST_CLASS_HID_MAIN_TAG_FEATURE	0xb
UX_HOST_CLASS_HID_MAIN_TAG_INPUT	0x8
UX_HOST_CLASS_HID_MAIN_TAG_OUTPUT	0x9
UX_HOST_CLASS_HID_MAX_CLIENTS	8
UX_HOST_CLASS_HID_MAX_COLLECTION	4
UX_HOST_CLASS_HID_MAX_REPORT	8
UX_HOST_CLASS_HID_MIN_MAX_ERROR	0x77
UX_HOST_CLASS_HID_PAGE_ALPHANUMERIC_DISPLAY	0x14
UX_HOST_CLASS_HID_PAGE_BAR_CODE_SCANNER	0x8C
UX_HOST_CLASS_HID_PAGE_BUTTON	0x09
UX_HOST_CLASS_HID_PAGE_CAMERA_CONTROL_PAGE	0x90
UX_HOST_CLASS_HID_PAGE_CONSUMER	0x0C
UX_HOST_CLASS_HID_PAGE_DIGITIZER	0x0D
UX_HOST_CLASS_HID_PAGE_GAME_CONTROLS	0x05
UX_HOST_CLASS_HID_PAGE_GENERIC_DESKTOP_CONTROLS	0x01
UX_HOST_CLASS_HID_PAGE_GENERIC_DEVICE_CONTROLS	0x06
UX_HOST_CLASS_HID_PAGE_KEYBOARD_KEYPAD	0x07
UX_HOST_CLASS_HID_PAGE_LEDS	0x08
UX_HOST_CLASS_HID_PAGE_MAGNETIC_STRIPE_READING	0x8E
UX_HOST_CLASS_HID_PAGE_MEDICAL_INSTRUMENTS	0x40
UX_HOST_CLASS_HID_PAGE_ORDINAL	0x0A
UX_HOST_CLASS_HID_PAGE_PHYSICAL_INTERFACE_DEVICE	0x0F
UX_HOST_CLASS_HID_PAGE_SCALE_PAGE	0x8D
UX_HOST_CLASS_HID_PAGE_SIMULATION_CONTROLS	0x02
UX_HOST_CLASS_HID_PAGE_SPORT_CONTROLS	0x04
UX_HOST_CLASS_HID_PAGE_TELEPHONY	0x0B
UX_HOST_CLASS_HID_PAGE_UNICODE	0x10

UX_HOST_CLASS_HID_PAGE_VR_CONTROLS	0x03
UX_HOST_CLASS_HID_PERIODIC_REPORT_ERROR	0x7A
UX_HOST_CLASS_HID_PHYSICAL_DESCRIPTOR	0x23
UX_HOST_CLASS_HID_POP_UNDERFLOW	0x74
UX_HOST_CLASS_HID_PUSH_OVERFLOW	0x73
UX_HOST_CLASS_HID_REPORT_DECOMPRESSED	0
UX_HOST_CLASS_HID_REPORT_DESCRIPTOR	0x22
UX_HOST_CLASS_HID_REPORT_ERROR	0x79
UX_HOST_CLASS_HID_REPORT_INDIVIDUAL_USAGE	2
UX_HOST_CLASS_HID_REPORT_OVERFLOW	0x70
UX_HOST_CLASS_HID_REPORT_RAW	1
UX_HOST_CLASS_HID_REPORT_SIZE	32
UX_HOST_CLASS_HID_REPORT_TYPE_FEATURE	0x3
UX_HOST_CLASS_HID_REPORT_TYPE_INPUT	0x1
UX_HOST_CLASS_HID_REPORT_TYPE_OUTPUT	0x2
UX_HOST_CLASS_HID_SET_IDLE	0x0A
UX_HOST_CLASS_HID_SET_PROTOCOL	0x0B
UX_HOST_CLASS_HID_SET_REPORT	0x09
UX_HOST_CLASS_HID_TAG_UNSUPPORTED	0x72
UX_HOST_CLASS_HID_TYPE_GLOBAL	0x1
UX_HOST_CLASS_HID_TYPE_LOCAL	0x2
UX_HOST_CLASS_HID_TYPE_MAIN	0x0
UX_HOST_CLASS_HID_TYPE_RESERVED	0x3
UX_HOST_CLASS_HID_UNKNOWN	0x7B
UX_HOST_CLASS_HID_USAGE_OVERFLOW	0x71
UX_HOST_CLASS_HUB_C_PORT_CONNECTION	0x10
UX_HOST_CLASS_HUB_C_PORT_ENABLE	0x11
UX_HOST_CLASS_HUB_C_PORT_OVER_CURRENT	0x13
UX_HOST_CLASS_HUB_C_PORT_RESET	0x14
UX_HOST_CLASS_HUB_C_PORT_SUSPEND	0x12
UX_HOST_CLASS_HUB_CLASS	9
UX_HOST_CLASS_HUB_CLEAR_FEATURE	0x01
UX_HOST_CLASS_HUB_COMPOUND_DEVICE	0x04
UX_HOST_CLASS_HUB_ENABLE_RETRY_COUNT	3
UX_HOST_CLASS_HUB_ENABLE_RETRY_DELAY	100
UX_HOST_CLASS_HUB_ENUMERATION_RETRY	3
UX_HOST_CLASS_HUB_ENUMERATION_RETRY_DELAY	300
UX_HOST_CLASS_HUB_GANG_POWER_SWITCHING	0x00

UX_HOST_CLASS_HUB_GET_DESCRIPTOR	0x06
UX_HOST_CLASS_HUB_GET_STATE	0x02
UX_HOST_CLASS_HUB_GET_STATUS	0x00
UX_HOST_CLASS_HUB_GLOBAL_OVERCURRENT	0x00
UX_HOST_CLASS_HUB_INDIVIDUAL_OVERCURRENT	0x08
UX_HOST_CLASS_HUB_INDIVIDUAL_POWER_SWITCHING	0x01
UX_HOST_CLASS_HUB_NO_OVERCURRENT	0x10
UX_HOST_CLASS_HUB_NO_POWER_SWITCHING	0x02
UX_HOST_CLASS_HUB_NOT_POWERED	8
UX_HOST_CLASS_HUB_PORT_CHANGE_CONNECTION	0x00001
UX_HOST_CLASS_HUB_PORT_CHANGE_ENABLE	0x00002
UX_HOST_CLASS_HUB_PORT_CHANGE_OVER_CURRENT	0x00008
UX_HOST_CLASS_HUB_PORT_CHANGE_RESET	0x00010
UX_HOST_CLASS_HUB_PORT_CHANGE_SUSPEND	0x00004
UX_HOST_CLASS_HUB_PORT_CONNECTION	0x00
UX_HOST_CLASS_HUB_PORT_ENABLE	0x01
UX_HOST_CLASS_HUB_PORT_LOW_SPEED	0x09
UX_HOST_CLASS_HUB_PORT_OVER_CURRENT	0x03
UX_HOST_CLASS_HUB_PORT_POWER	0x08
UX_HOST_CLASS_HUB_PORT_RESET	0x04
UX_HOST_CLASS_HUB_PORT_STATUS_CONNECTION	0x0001
UX_HOST_CLASS_HUB_PORT_STATUS_ENABLE	0x0002
UX_HOST_CLASS_HUB_PORT_STATUS_HIGH_SPEED	0x0400
UX_HOST_CLASS_HUB_PORT_STATUS_LOW_SPEED	0x0200
UX_HOST_CLASS_HUB_PORT_STATUS_OVER_CURRENT	0x0008
UX_HOST_CLASS_HUB_PORT_STATUS_POWER	0x0100
UX_HOST_CLASS_HUB_PORT_STATUS_RESET	0x0010
UX_HOST_CLASS_HUB_PORT_STATUS_SUSPEND	0x0004
UX_HOST_CLASS_HUB_PORT_SUSPEND	0x02
UX_HOST_CLASS_HUB_PROTOCOL_FS	0
UX_HOST_CLASS_HUB_PROTOCOL_MULTIPLE_TT	2
UX_HOST_CLASS_HUB_PROTOCOL_SINGLE_TT	1
UX_HOST_CLASS_HUB_SET_DESCRIPTOR	0x07
UX_HOST_CLASS_HUB_SET_FEATURE	0x03
UX_HOST_CLASS_INSTANCE_FREE	0
UX_HOST_CLASS_INSTANCE_LIVE	1
UX_HOST_CLASS_INSTANCE_OPENED	3
UX_HOST_CLASS_INSTANCE_SHUTDOWN	2

UX_HOST_CLASS_INSTANCE_UNKNOWN	0x5b
UX_HOST_CLASS_MEDIA_NOT_SUPPORTED	0x62
UX_HOST_CLASS_MEMORY_ERROR	0x61
UX_HOST_CLASS_PRINTER_CLASS	7
UX_HOST_CLASS_PRINTER_CLASS_TRANSFER_TIMEOUT	300000
UX_HOST_CLASS_PRINTER_DESCRIPTOR_LENGTH	1024
UX_HOST_CLASS_PRINTER_GENERIC_NAME	"USB PRINTER"
UX_HOST_CLASS_PRINTER_GET_DEVICE_ID	0
UX_HOST_CLASS_PRINTER_GET_STATUS	1
UX_HOST_CLASS_PRINTER_NAME_LENGTH	64
UX_HOST_CLASS_PRINTER_NAME_LENGTH	64
UX_HOST_CLASS_PRINTER_PROTOCOL_BI_DIRECTIONAL	2
UX_HOST_CLASS_PRINTER_SOFT_RESET	2
UX_HOST_CLASS_PRINTER_STATUS_LENGTH	4
UX_HOST_CLASS_PRINTER_SUBCLASS	1
UX_HOST_CLASS_PRINTER_TAG_DES	"DES:"
UX_HOST_CLASS_PRINTER_TAG_DESCRIPTION	"DESCRIPTION:"
UX_HOST_CLASS_PROTOCOL_ERROR	0x60
UX_HOST_CLASS_STORAGE_CBI_STATUS_TIMEOUT	3000
UX_HOST_CLASS_STORAGE_CBW_CB	15
UX_HOST_CLASS_STORAGE_CBW_CB_LENGTH	14
UX_HOST_CLASS_STORAGE_CBW_DATA_LENGTH	8
UX_HOST_CLASS_STORAGE_CBW_FLAGS	12
UX_HOST_CLASS_STORAGE_CBW_LUN	13
UX_HOST_CLASS_STORAGE_CBW_SIGNATURE	0
UX_HOST_CLASS_STORAGE_CBW_SIGNATURE_MASK	0x43425355
UX_HOST_CLASS_STORAGE_CBW_SIZE	64
UX_HOST_CLASS_STORAGE_CBW_TAG	4
UX_HOST_CLASS_STORAGE_CBW_TAG_MASK	0x55534243
UX_HOST_CLASS_STORAGE_CLASS	8
UX_HOST_CLASS_STORAGE_COMMAND_ERROR	2
UX_HOST_CLASS_STORAGE_CSW_DATA_RESIDUE	8
UX_HOST_CLASS_STORAGE_CSW_FAILED	13
UX_HOST_CLASS_STORAGE_CSW_PASSED	0
UX_HOST_CLASS_STORAGE_CSW_PHASE_ERROR	2
UX_HOST_CLASS_STORAGE_CSW_SIGNATURE	0
UX_HOST_CLASS_STORAGE_CSW_STATUS	12
UX_HOST_CLASS_STORAGE_CSW_TAG	4

UX_HOST_CLASS_STORAGE_DATA_IN	0x80
UX_HOST_CLASS_STORAGE_DATA_OUT	0
UX_HOST_CLASS_STORAGE_DEVICE_INIT_DELAY	(2 * UX_PERIODIC_RATE)
UX_HOST_CLASS_STORAGE_ERROR_MEDIA_NOT_READ	0x023A00
UX_HOST_CLASS_STORAGE_GET_MAX_LUN	0xfe
UX_HOST_CLASS_STORAGE_INQUIRY_ALLOCATION_LENGTH	4
UX_HOST_CLASS_STORAGE_INQUIRY_COMMAND_LENGTH_SBC	06
UX_HOST_CLASS_STORAGE_INQUIRY_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_INQUIRY_LUN	1
UX_HOST_CLASS_STORAGE_INQUIRY_OPERATION	0
UX_HOST_CLASS_STORAGE_INQUIRY_PAGE_CODE	2
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_ADDITIONAL_LENGTH	4
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_DATA_FORMAT	3
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_LENGTH	36
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_PERIPHERAL_TYPE	0
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_PRODUCT_ID	16
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_PRODUCT_REVISION	32
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_REMOVABLE_MEDIA	1
UX_HOST_CLASS_STORAGE_INQUIRY_RESPONSE_VENDOR_INFORMATION	8
UX_HOST_CLASS_STORAGE_MAX_MEDIA	8
UX_HOST_CLASS_STORAGE_MEDIA_CDROM	5
UX_HOST_CLASS_STORAGE_MEDIA_FAT_DISK	0
UX_HOST_CLASS_STORAGE_MEDIA_IOMEGA_CLICK	0x55
UX_HOST_CLASS_STORAGE_MEDIA_KNOWN	1
UX_HOST_CLASS_STORAGE_MEDIA_MOUNTED	1
UX_HOST_CLASS_STORAGE_MEDIA_NAME	"usb disk"
UX_HOST_CLASS_STORAGE_MEDIA_OPTICAL_DISK	7
UX_HOST_CLASS_STORAGE_MEDIA_REMOVABLE	0x80
UX_HOST_CLASS_STORAGE_MEDIA_UNKNOWN	0
UX_HOST_CLASS_STORAGE_MEDIA_UNMOUNTED	0
UX_HOST_CLASS_STORAGE_MEMORY_BUFFER_SIZE	1024*8
UX_HOST_CLASS_STORAGE_MODE_SENSE_ALL_PAGE	0x3F
UX_HOST_CLASS_STORAGE_MODE_SENSE_ALL_PAGE_LENGTH	0xC0
UX_HOST_CLASS_STORAGE_MODE_SENSE_COMMAND_LENGTH_SBC	12
UX_HOST_CLASS_STORAGE_MODE_SENSE_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_MODE_SENSE_FD_PAGE	0x05
UX_HOST_CLASS_STORAGE_MODE_SENSE_FD_PAGE_LENGTH	0x20
UX_HOST_CLASS_STORAGE_MODE_SENSE_HEADER_PAGE_LENGTH	0x08

UX_HOST_CLASS_STORAGE_MODE_SENSE_LUN	1
UX_HOST_CLASS_STORAGE_MODE_SENSE_OPERATION	0
UX_HOST_CLASS_STORAGE_MODE_SENSE_PARAMETER_LIST_LENGTH	7
UX_HOST_CLASS_STORAGE_MODE_SENSE_PC_PAGE_CODE	2
UX_HOST_CLASS_STORAGE_MODE_SENSE_RBAC_PAGE	0x1B
UX_HOST_CLASS_STORAGE_MODE_SENSE_RBAC_PAGE_LENGTH	0x0c
UX_HOST_CLASS_STORAGE_MODE_SENSE_RESPONSE_ATTRIBUTES	3
UX_HOST_CLASS_STORAGE_MODE_SENSE_RESPONSE_ATTRIBUTES_SHORT	2
UX_HOST_CLASS_STORAGE_MODE_SENSE_RESPONSE_ATTRIBUTES_WP	0x80
UX_HOST_CLASS_STORAGE_MODE_SENSE_RESPONSE_MEDIUM_TYPE_CODE	2
UX_HOST_CLASS_STORAGE_MODE_SENSE_RESPONSE_MODE_DATA_LENGTH	0
UX_HOST_CLASS_STORAGE_MODE_SENSE_RWER_PAGE	0x01
UX_HOST_CLASS_STORAGE_MODE_SENSE_RWER_PAGE_LENGTH	0x0c
UX_HOST_CLASS_STORAGE_MODE_SENSE_TP_PAGE	0x1C
UX_HOST_CLASS_STORAGE_MODE_SENSE_TP_PAGE_LENGTH	0x08
UX_HOST_CLASS_STORAGE_PARTITION_BOOT_FLAG	0
UX_HOST_CLASS_STORAGE_PARTITION_END_HEAD	5
UX_HOST_CLASS_STORAGE_PARTITION_END_SECTOR	6
UX_HOST_CLASS_STORAGE_PARTITION_END_TRACK	7
UX_HOST_CLASS_STORAGE_PARTITION_EXTENDED	5
UX_HOST_CLASS_STORAGE_PARTITION_EXTENDED_LBA_MAPPED	0x0f
UX_HOST_CLASS_STORAGE_PARTITION_FAT_12	1
UX_HOST_CLASS_STORAGE_PARTITION_FAT_16	4
UX_HOST_CLASS_STORAGE_PARTITION_FAT_16_LBA_MAPPED	0x0e
UX_HOST_CLASS_STORAGE_PARTITION_FAT_16L	6
UX_HOST_CLASS_STORAGE_PARTITION_FAT_32_1	0x0b
UX_HOST_CLASS_STORAGE_PARTITION_FAT_32_2	0x0c
UX_HOST_CLASS_STORAGE_PARTITION_NUMBER_SECTORS	12
UX_HOST_CLASS_STORAGE_PARTITION_SECTORS_BEFORE	8
UX_HOST_CLASS_STORAGE_PARTITION_SIGNATURE	0xaa55
UX_HOST_CLASS_STORAGE_PARTITION_START_HEAD	1
UX_HOST_CLASS_STORAGE_PARTITION_START_SECTOR	2
UX_HOST_CLASS_STORAGE_PARTITION_START_16	
UX_HOST_CLASS_STORAGE_PARTITION_TABLE_START	446
UX_HOST_CLASS_STORAGE_PARTITION_TYPE	4
UX_HOST_CLASS_STORAGE_PROTOCOL_BO	0x50
UX_HOST_CLASS_STORAGE_PROTOCOL_CB	1
UX_HOST_CLASS_STORAGE_PROTOCOL_CBI	0

UX_HOST_CLASS_STORAGE_READ_CAPACITY_COMMAND_LENGTH_SBC	10
UX_HOST_CLASS_STORAGE_READ_CAPACITY_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_READ_CAPACITY_DATA_LBA	0
UX_HOST_CLASS_STORAGE_READ_CAPACITY_DATA_SECTOR_SIZE	4
UX_HOST_CLASS_STORAGE_READ_CAPACITY_LBA	2
UX_HOST_CLASS_STORAGE_READ_CAPACITY_LUN	1
UX_HOST_CLASS_STORAGE_READ_CAPACITY_OPERATION	0
UX_HOST_CLASS_STORAGE_READ_CAPACITY_RESPONSE_LENGTH	8
UX_HOST_CLASS_STORAGE_READ_COMMAND_LENGTH_SBC	10
UX_HOST_CLASS_STORAGE_READ_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_READ_FORMAT_COMMAND_LENGTH_SBC	10
UX_HOST_CLASS_STORAGE_READ_FORMAT_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_READ_FORMAT_LBA	2
UX_HOST_CLASS_STORAGE_READ_FORMAT_LUN	1
UX_HOST_CLASS_STORAGE_READ_FORMAT_OPERATION	0
UX_HOST_CLASS_STORAGE_READ_FORMAT_PARAMETER_LIST_LENGTH	7
UX_HOST_CLASS_STORAGE_READ_FORMAT_RESPONSE_LENGTH	0xFC
UX_HOST_CLASS_STORAGE_READ_LBA	2
UX_HOST_CLASS_STORAGE_READ_LUN	1
UX_HOST_CLASS_STORAGE_READ_OPERATION	0
UX_HOST_CLASS_STORAGE_READ_TRANSFER_LENGTH	7
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_ALLOCATION_LENGTH	4
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_COMMAND_LENGTH_SBC	12
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_LUN	1
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_OPERATION	0
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_ADD_LENGTH	7
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_CODE	12
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_CODE_QUALIFIER	13
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_ERROR_CODE	0
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_INFORMATION	3
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_LENGTH	18
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_SENSE_KEY	2
UX_HOST_CLASS_STORAGE_REQUEST_SENSE_RETRY	10
UX_HOST_CLASS_STORAGE_RESET	0xff
UX_HOST_CLASS_STORAGE_SCSI_FORMAT	0x04
UX_HOST_CLASS_STORAGE_SCSI_INQUIRY	0x12
UX_HOST_CLASS_STORAGE_SCSI_MODE_SELECT	0x55

UX_HOST_CLASS_STORAGE_SCSI_MODE_SENSE	0x5a
UX_HOST_CLASS_STORAGE_SCSI_MODE_SENSE_SHORT	0x1a
UX_HOST_CLASS_STORAGE_SCSI_READ_CAPACITY	0x25
UX_HOST_CLASS_STORAGE_SCSI_READ_FORMAT_CAPACITY	0x23
UX_HOST_CLASS_STORAGE_SCSI_READ16	0x28
UX_HOST_CLASS_STORAGE_SCSI_READ32	0xa8
UX_HOST_CLASS_STORAGE_SCSI_REQUEST_SENSE	0x03
UX_HOST_CLASS_STORAGE_SCSI_START_STOP	0x1b
UX_HOST_CLASS_STORAGE_SCSI_TEST_READY	0x00
UX_HOST_CLASS_STORAGE_SCSI_VERIFY	0x2f
UX_HOST_CLASS_STORAGE_SCSI_WRITE16	0x2a
UX_HOST_CLASS_STORAGE_SCSI_WRITE32	0xaa
UX_HOST_CLASS_STORAGE_SECTOR_SIZE_FAT	512
UX_HOST_CLASS_STORAGE_SECTOR_SIZE_OTHER	2048
UX_HOST_CLASS_STORAGE_SENSE_ERROR	3
UX_HOST_CLASS_STORAGE_SENSE_KEY_ABORTED_COMMAND	0x0b
UX_HOST_CLASS_STORAGE_SENSE_KEY_BLANK_CHECK	0x8
UX_HOST_CLASS_STORAGE_SENSE_KEY_DATA_PROTECT	0x7
UX_HOST_CLASS_STORAGE_SENSE_KEY_HARDWARE_ERROR	0x4
UX_HOST_CLASS_STORAGE_SENSE_KEY_ILLEGAL_REQUEST	0x5
UX_HOST_CLASS_STORAGE_SENSE_KEY_MEDIUM_ERROR	0x3
UX_HOST_CLASS_STORAGE_SENSE_KEY_MISCOMPARE	0x0e
UX_HOST_CLASS_STORAGE_SENSE_KEY_NO_SENSE	0x0
UX_HOST_CLASS_STORAGE_SENSE_KEY_NOT_READY	0x2
UX_HOST_CLASS_STORAGE_SENSE_KEY_RECOVERED_ERROR	0x1
UX_HOST_CLASS_STORAGE_SENSE_KEY_UNIT_ATTENTION	0x0d
UX_HOST_CLASS_STORAGE_START_MEDIA	1
UX_HOST_CLASS_STORAGE_START_STOP_COMMAND_LENGTH_SBC	12
UX_HOST_CLASS_STORAGE_START_STOP_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_START_STOP_LBUFLAGS	1
UX_HOST_CLASS_STORAGE_START_STOP_OPERATION	0
UX_HOST_CLASS_STORAGE_START_STOP_START_BIT	4
UX_HOST_CLASS_STORAGE_STOP_MEDIA	0
UX_HOST_CLASS_STORAGE_SUBCLASS_RBC	1
UX_HOST_CLASS_STORAGE_SUBCLASS_SCSI	6
UX_HOST_CLASS_STORAGE_SUBCLASS_SFF8020	2
UX_HOST_CLASS_STORAGE_SUBCLASS_SFF8070	5
UX_HOST_CLASS_STORAGE_SUBCLASS_UFI	4

UX_HOST_CLASS_STORAGE_TEST_READY_COMMAND_LENGTH_SBC	6
UX_HOST_CLASS_STORAGE_TEST_READY_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_TEST_READY_LUN	1
UX_HOST_CLASS_STORAGE_TEST_READY_OPERATION	0
UX_HOST_CLASS_STORAGE_THREAD_PRIORITY_CLASS	20
UX_HOST_CLASS_STORAGE_THREAD_SLEEP_TIME	(2 * UX_PERIODIC_RATE)
UX_HOST_CLASS_STORAGE_THREAD_STACK_SIZE	4096
UX_HOST_CLASS_STORAGE_TRANSFER_TIMEOUT	10000
UX_HOST_CLASS_STORAGE_TRANSPORT_ERROR	1
UX_HOST_CLASS_STORAGE_WRITE_COMMAND_LENGTH_SBC	10
UX_HOST_CLASS_STORAGE_WRITE_COMMAND_LENGTH_UFI	12
UX_HOST_CLASS_STORAGE_WRITE_LBA	2
UX_HOST_CLASS_STORAGE_WRITE_LUN	1
UX_HOST_CLASS_STORAGE_WRITE_OPERATION	0
UX_HOST_CLASS_STORAGE_WRITE_TRANSFER_LENGTH	7
UX_HOST_CLASS_UNKNOWN	0x59
UX_HUB_DESCRIPTOR_ENTRIES	8
UX_HUB_DESCRIPTOR_LENGTH	9
UX_INTERFACE_DESCRIPTOR_ITEM	4
UX_INTERFACE_HANDLE_UNKNOWN	0x52
UX_INTERRUPT_ENDPOINT	3
UX_INTERRUPT_ENDPOINT_IN	0x83
UX_INTERRUPT_ENDPOINT_OUT	0x03
UX_ISOCHRONOUS_ENDPOINT	1
UX_ISOCHRONOUS_ENDPOINT_IN	0x81
UX_ISOCHRONOUS_ENDPOINT_OUT	0x01
UX_LOW_SPEED_DEVICE	0
UX_MASK_ENDPOINT_TYPE	3
UX_MAX_BUS_POWER	(100/2)
UX_MAX_CLASS_DRIVER	8
UX_MAX_DEVICES	8
UX_MAX_ED	80
UX_MAX_HCD	1
UX_MAX_ISO_TD	128
UX_MAX_SELF_POWER	(500/2)
UX_MAX_SLAVE_CLASS_DRIVER	3
UX_MAX_SLAVE_LUN	2
UX_MAX_TD	32

UX_MAX_TT	8
UX_MAX_USB_DEVICES	127
UX_MEMORY_INSUFFICIENT	0x12
UX_MEMORY_UNUSED	0x12345678
UX_MEMORY_USED	0x87654321
UX_MUTEX_ERROR	0x17
UX_NO_ALIGN	0
UX_NO_ALTERNATE_SETTING	0x5e
UX_NO_BANDWIDTH_AVAILABLE	0x41
UX_NO_CLASS_MATCH	0x57
UX_NO_ED_AVAILABLE	0x14
UX_NO_TD_AVAILABLE	0x13
UX_NO_TIME_SLICE	0
UX_NULL	((void*)0)
UX_OTHER_SPEED_DESCRIPTOR_ITEM	7
UX_OVER_CURRENT_CONDITION	0x43
UX_PCI_CFG_BASE_ADDRESS_0	0x10
UX_PCI_CFG_BASE_ADDRESS_1	0x14
UX_PCI_CFG_BASE_ADDRESS_2	0x18
UX_PCI_CFG_BASE_ADDRESS_3	0x1c
UX_PCI_CFG_BASE_ADDRESS_4	0x20
UX_PCI_CFG_BASE_ADDRESS_5	0x24
UX_PCI_CFG_BIST	0x0f
UX_PCI_CFG_CACHE_LINE_SIZE	0x0c
UX_PCI_CFG_CARDBUS_CIS	0x28
UX_PCI_CFG_CLASS	0x0b
UX_PCI_CFG_COMMAND	0x04
UX_PCI_CFG_CTRL_ADDRESS	0x0cf8
UX_PCI_CFG_DATA_ADDRESS	0x0cfc
UX_PCI_CFG_DEVICE_ID	0x02
UX_PCI_CFG_EXPANSION_ROM_ADDRESS	0x30
UX_PCI_CFG_FLADJ	0x61
UX_PCI_CFG_HEADER_TYPE	0x0e
UX_PCI_CFG_INT_LINE	0x3c
UX_PCI_CFG_INT_PIN	0x3d
UX_PCI_CFG_LATENCY_TIMER	0x0d
UX_PCI_CFG_MAX_LATENCY	0x3f
UX_PCI_CFG_MIN_GNT	0x3e

UX_PCI_CFG_PROGRAMMING_IF	0x09
UX_PCI_CFG_RESERVED_0	0x34
UX_PCI_CFG_RESERVED_1	0x38
UX_PCI_CFG_REVISION	0x08
UX_PCI_CFG_SBRN	0x60
UX_PCI_CFG_STATUS	0x06
UX_PCI_CFG_SUB_SYSTEM_ID	0x2e
UX_PCI_CFG_SUB_VENDOR_ID	0x2c
UX_PCI_CFG_SUBCLASS	0x0a
UX_PCI_CFG_VENDOR_ID	0x00
UX_PCI_CMD_FBB_ENABLE	0x0200
UX_PCI_CMD_IO_ENABLE	0x0001
UX_PCI_CMD_MASTER_ENABLE	0x0004
UX_PCI_CMD_MEM_ENABLE	0x0002
UX_PCI_CMD_MEM_WRITE_INV_ENABLE	0x0010
UX_PCI_CMD_MONITOR_ENABLE	0x0008
UX_PCI_CMD_PARITY_ERROR_ENABLE	0x0040
UX_PCI_CMD_SERR_ENABLE	0x0100
UX_PCI_CMD_SNOOP_PALETTE_ENABLE	0x0020
UX_PCI_CMD_WAIT_CYCLE_CTRL_ENABLE	0x0080
UX_PCI_NB_BUS	0xff
UX_PCI_NB_DEVICE	32
UX_PCI_NB_FUNCTIONS	7
UX_PERIODIC_RATE	1000
UX_PORT_ENABLE_WAIT	50
UX_PORT_INDEX_UNKNOWN	0x56
UX_PORT_RESET_FAILED	0x31
UX_PS_CCS	0x01
UX_PS_CPE	0x01
UX_PS_DS	6
UX_PS_DS_FS	0x40
UX_PS_DS_HS	0x80
UX_PS_DS_LS	0x00
UX_PS_PES	0x02
UX_PS_POCI	0x08
UX_PS_PPS	0x20
UX_PS_PRS	0x10
UX_PS_PSS	0x04

UX_REGULAR_MEMORY	0
UX_REQUEST_DIRECTION	0x80
UX_REQUEST_IN	0x80
UX_REQUEST_OUT	0x00
UX_REQUEST_TARGET	0x03
UX_REQUEST_TARGET_DEVICE	0x00
UX_REQUEST_TARGET_ENDPOINT	0x02
UX_REQUEST_TARGET_INTERFACE	0x01
UX_REQUEST_TARGET_OTHER	0x03
UX_REQUEST_TYPE	0x60
UX_REQUEST_TYPE_CLASS	0x20
UX_REQUEST_TYPE_STANDARD	0x00
UX_REQUEST_TYPE_VENDOR	0x40
UX_RH_ENUMERATION_RETRY	3
UX_RH_ENUMERATION_RETRY_DELAY	100
UX_SEMAPHORE_ERROR	0x15
UX_SET_ADDRESS	5
UX_SET_CONFIGURATION	9
UX_SET_DESCRIPTOR	7
UX_SET_FEATURE	3
UX_SET_INTERFACE	11
UX_SETUP_INDEX	4
UX_SETUP_LENGTH	6
UX_SETUP_REQUEST	1
UX_SETUP_REQUEST_TYPE	0
UX_SETUP_SIZE	8
UX_SETUP_VALUE	2
UX_SLAVE_CLASS_COMMAND_ACTIVATE	2
UX_SLAVE_CLASS_COMMAND_DEACTIVATE	3
UX_SLAVE_CLASS_COMMAND_QUERY	1
UX_SLAVE_CLASS_COMMAND_REQUEST	4
UX_SLAVE_CLASS_STORAGE_ASC_KEY_INVALID_COMMAND	0x20
UX_SLAVE_CLASS_STORAGE_BUFFER_SIZE UX_SLAVE_REQUEST_DATA_MAX_LENGTH	
UX_SLAVE_CLASS_STORAGE_CBW_CB	15
UX_SLAVE_CLASS_STORAGE_CBW_CB_LENGTH	14
UX_SLAVE_CLASS_STORAGE_CBW_DATA_LENGTH	8
UX_SLAVE_CLASS_STORAGE_CBW_FLAGS	12

UX_SLAVE_CLASS_STORAGE_CBW_LUN	13
UX_SLAVE_CLASS_STORAGE_CBW_SIGNATURE	0
UX_SLAVE_CLASS_STORAGE_CBW_SIGNATURE_MASK	0x43425355
UX_SLAVE_CLASS_STORAGE_CBW_TAG	4
UX_SLAVE_CLASS_STORAGE_CLASS	8
UX_SLAVE_CLASS_STORAGE_CSW_DATA_RESIDUE	8
UX_SLAVE_CLASS_STORAGE_CSW_FAILED	1
UX_SLAVE_CLASS_STORAGE_CSW_LENGTH	13
UX_SLAVE_CLASS_STORAGE_CSW_PASSED	0
UX_SLAVE_CLASS_STORAGE_CSW_PHASE_ERROR	2
UX_SLAVE_CLASS_STORAGE_CSW_SIGNATURE	0
UX_SLAVE_CLASS_STORAGE_CSW_SIGNATURE_MASK	0x53425355
UX_SLAVE_CLASS_STORAGE_CSW_STATUS	12
UX_SLAVE_CLASS_STORAGE_CSW_TAG	4
UX_SLAVE_CLASS_STORAGE_GET_MAX_LUN	0xfe
UX_SLAVE_CLASS_STORAGE_INQUIRY_ALLOCATION_LENGTH	4
UX_SLAVE_CLASS_STORAGE_INQUIRY_COMMAND_LENGTH_SBC	06
UX_SLAVE_CLASS_STORAGE_INQUIRY_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_INQUIRY_LUN	1
UX_SLAVE_CLASS_STORAGE_INQUIRY_OPERATION	0
UX_SLAVE_CLASS_STORAGE_INQUIRY_PAGE_CODE	2
UX_SLAVE_CLASS_STORAGE_INQUIRY_PAGE_CODE_STANDARD	0x00
UX_SLAVE_CLASS_STORAGE_INQUIRY_PERIPHERAL_TYPE	0x00
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_ADDITIONAL_LENGTH	4
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_DATA_FORMAT	3
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_LENGTH	36
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_LENGTH_CD_ROM	0x5b
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_PERIPHERAL_TYPE	0
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_PRODUCT_ID	16
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_PRODUCT_REVISION	32
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_REMOVABLE_MEDIA	1
UX_SLAVE_CLASS_STORAGE_INQUIRY_RESPONSE_VENDOR_INFORMATION	8
UX_SLAVE_CLASS_STORAGE_MAX_LUN	0
UX_SLAVE_CLASS_STORAGE_MEDIA_CDROM	5
UX_SLAVE_CLASS_STORAGE_MEDIA_FAT_DISK	0
UX_SLAVE_CLASS_STORAGE_MEDIA_IOMEGA_CLICK	0x55
UX_SLAVE_CLASS_STORAGE_MEDIA_OPTICAL_DISK	7
UX_SLAVE_CLASS_STORAGE_MODE_SENSE_COMMAND_LENGTH_SBC	12

UX_SLAVE_CLASS_STORAGE_MODE_SENSE_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_MODE_SENSE_LUN	1
UX_SLAVE_CLASS_STORAGE_MODE_SENSE_OPERATION	0
UX_SLAVE_CLASS_STORAGE_MODE_SENSE_PARAMETER_LIST_LENGTH	7
UX_SLAVE_CLASS_STORAGE_MODE_SENSE_PC_PAGE_CODE	2
UX_SLAVE_CLASS_STORAGE_PROTOCOL_BO	0x50
UX_SLAVE_CLASS_STORAGE_PROTOCOL_CB	1
UX_SLAVE_CLASS_STORAGE_PROTOCOL_CBI	0
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_COMMAND_LENGTH_SBC	10
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_LBA	2
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_LUN	1
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_OPERATION	0
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_RESPONSE_BLOCK_SIZE	4
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_RESPONSE_LAST_LBA	0
UX_SLAVE_CLASS_STORAGE_READ_CAPACITY_RESPONSE_LENGTH	8
UX_SLAVE_CLASS_STORAGE_READ_COMMAND_LENGTH_SBC	10
UX_SLAVE_CLASS_STORAGE_READ_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_READ_LBA	2
UX_SLAVE_CLASS_STORAGE_READ_LUN	1
UX_SLAVE_CLASS_STORAGE_READ_OPERATION	0
UX_SLAVE_CLASS_STORAGE_READ_TRANSFER_LENGTH_16	7
UX_SLAVE_CLASS_STORAGE_READ_TRANSFER_LENGTH_32	6
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_ALLOCATION_LENGTH	4
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_COMMAND_LENGTH_SBC	12
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_LUN	1
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_OPERATION	0
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_ADD_LENGTH	7
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_CODE	12
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_CODE_QUALIFIER	13
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_ERROR_CODE	0
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_ERROR_CODE_VALUE	0x70
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_INFORMATION	3
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_LENGTH	18
UX_SLAVE_CLASS_STORAGE_REQUEST_SENSE_RESPONSE_SENSE_KEY	2
UX_SLAVE_CLASS_STORAGE_RESET	0xff
UX_SLAVE_CLASS_STORAGE_RESPONSE_LENGTH	64

UX_SLAVE_CLASS_STORAGE_SCSI_FORMAT	0x04
UX_SLAVE_CLASS_STORAGE_SCSI_INQUIRY	0x12
UX_SLAVE_CLASS_STORAGE_SCSI_MODE_SELECT	0x55
UX_SLAVE_CLASS_STORAGE_SCSI_MODE_SENSE	0x5a
UX_SLAVE_CLASS_STORAGE_SCSI_MODE_SENSE_SHORT	0x1a
UX_SLAVE_CLASS_STORAGE_SCSI_PREVENT_ALLOW_MEDIA_REMOVAL	0x1e
UX_SLAVE_CLASS_STORAGE_SCSI_READ_CAPACITY	0x25
UX_SLAVE_CLASS_STORAGE_SCSI_READ_FORMAT_CAPACITY	0x23
UX_SLAVE_CLASS_STORAGE_SCSI_READ_TOC	0x43
UX_SLAVE_CLASS_STORAGE_SCSI_READ16	0x28
UX_SLAVE_CLASS_STORAGE_SCSI_READ32	0xa8
UX_SLAVE_CLASS_STORAGE_SCSI_REQUEST_SENSE	0x03
UX_SLAVE_CLASS_STORAGE_SCSI_START_STOP	0x1b
UX_SLAVE_CLASS_STORAGE_SCSI_TEST_READY	0x00
UX_SLAVE_CLASS_STORAGE_SCSI_VERIFY	0x2f
UX_SLAVE_CLASS_STORAGE_SCSI_WRITE16	0x2a
UX_SLAVE_CLASS_STORAGE_SCSI_WRITE32	0xaa
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_ABORTED_COMMAND	0x0b
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_BLANK_CHECK	0x8
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_DATA_PROTECT	0x7
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_HARDWARE_ERROR	0x4
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_ILLEGAL_REQUEST	0x5
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_MEDIUM_ERROR	0x3
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_MISCOMPARE	0x0e
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_NO_SENSE	0x0
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_NOT_READY	0x2
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_RECOVERED_ERROR	0x1
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_UNIT_ATTENTION	0x6
UX_SLAVE_CLASS_STORAGE_SENSE_KEY_VOLUME_OVERFLOW	0x0d
UX_SLAVE_CLASS_STORAGE_START_STOP_COMMAND_LENGTH_SBC	6
UX_SLAVE_CLASS_STORAGE_START_STOP_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_START_STOP_LBUFLAGS	1
UX_SLAVE_CLASS_STORAGE_START_STOP_OPERATION	0
UX_SLAVE_CLASS_STORAGE_START_STOP_START_BIT	4
UX_SLAVE_CLASS_STORAGE_SUBCLASS_RBC	1
UX_SLAVE_CLASS_STORAGE_SUBCLASS_SCSI	6
UX_SLAVE_CLASS_STORAGE_SUBCLASS_SFF8020	2
UX_SLAVE_CLASS_STORAGE_SUBCLASS_SFF8070	5

UX_SLAVE_CLASS_STORAGE_SUBCLASS_UFI	4
UX_SLAVE_CLASS_STORAGE_TEST_READY_COMMAND_LENGTH_SBC	6
UX_SLAVE_CLASS_STORAGE_TEST_READY_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_TEST_READY_LUN	1
UX_SLAVE_CLASS_STORAGE_TEST_READY_OPERATION	0
UX_SLAVE_CLASS_STORAGE_WRITE_COMMAND_LENGTH_SBC	10
UX_SLAVE_CLASS_STORAGE_WRITE_COMMAND_LENGTH_UFI	12
UX_SLAVE_CLASS_STORAGE_WRITE_LBA	2
UX_SLAVE_CLASS_STORAGE_WRITE_LUN	1
UX_SLAVE_CLASS_STORAGE_WRITE_OPERATION	0
UX_SLAVE_CLASS_STORAGE_WRITE_TRANSFER_LENGTH_16	7
UX_SLAVE_CLASS_STORAGE_WRITE_TRANSFER_LENGTH_32	6
UX_SLAVE_ENDPOINT_DEFAULT_BUFFER_SIZE	256
UX_SLAVE_REQUEST_CONTROL_MAX_LENGTH	256
UX_SLAVE_REQUEST_DATA_MAX_LENGTH	4096
UX_STATUS_DEVICE_SELF_POWERED	1
UX_STRING_DESCRIPTOR_ITEM	3
UX_SUCCESS	0
UX_SYNCH_FRAME	12
UX_THREAD_ERROR	0x16
UX_THREAD_PRIORITY_CLASS	20
UX_THREAD_PRIORITY_DCD	2
UX_THREAD_PRIORITY_ENUM	20
UX_THREAD_PRIORITY_HCD	2
UX_THREAD_PRIORITY_KEYBOARD	20
UX_THREAD_STACK_SIZE	1024
UX_TOO_MANY_DEVICES	0x11
UX_TRANSFER_BUFFER_OVERFLOW	0x27
UX_TRANSFER_BUS_RESET	0x26
UX_TRANSFER_ERROR	0x23
UX_TRANSFER_MISSED_FRAME	0x24
UX_TRANSFER_NO_ANSWER	0x22
UX_TRANSFER_NOT_READY	0x25
UX_TRANSFER_PHASE_DATA_IN	2
UX_TRANSFER_PHASE_DATA_OUT	3
UX_TRANSFER_PHASE_SETUP	1
UX_TRANSFER_PHASE_STATUS_IN	4
UX_TRANSFER_PHASE_STATUS_OUT	5

UX_TRANSFER_STALLED	0x21
UX_TRANSFER_STATUS_ABORT	4
UX_TRANSFER_STATUS_COMPLETED	2
UX_TRANSFER_STATUS_NOT_PENDING	0
UX_TRANSFER_STATUS_PENDING	1
UX_TRANSFER_TIMEOUT	0x5c
UX_TRUE	1
UX_TT_BANDWIDTH	900
UX_TT_MASK	0x1f7
UX_UNUSED	0
UX_USED	1
UX_WAIT_FOREVER	0xffffffff

U S B X

APPENDIX C

USBX Data Types

USBX HUB Data Types

```
typedef struct UX_HUB_TT_STRUCT
{

    ULONG           ux_hub_tt_port_mapping;
    ULONG           ux_hub_tt_max_bandwidth;
} UX_HUB_TT;


typedef struct UX_HOST_CLASS_COMMAND_STRUCT
{

    UINT            ux_host_class_command_request;
    VOID            *ux_host_class_command_container;
    VOID            *ux_host_class_command_instance;
    UINT            ux_host_class_command_usage;
    UINT            ux_host_class_command_pid;
    UINT            ux_host_class_command_vid;
    UINT            ux_host_class_command_class;
    UINT            ux_host_class_command_subclass;
    UINT            ux_host_class_command_protocol;
    struct UX_HOST_CLASS_STRUCT
                    *ux_host_class_command_class_ptr;
} UX_HOST_CLASS_COMMAND;
```

USBX Host Stack Data Types

```
typedef struct UX_HOST_CLASS_STRUCT
{

    UCHAR           ux_host_class_name[32];
    UINT            ux_host_class_status;
    UINT            (*ux_host_class_entry_function)
                                (struct UX_HOST_CLASS_COMMAND_STRUCT *) ;
    UINT            ux_host_class_nb_devices_owned;
    VOID            *ux_host_class_first_instance;
    VOID            *ux_host_class_client;
    TX_THREAD       ux_host_class_thread;
    VOID            *ux_host_class_thread_stack;
    VOID            *ux_host_class_media;
} UX_HOST_CLASS;


typedef struct UX_TRANSFER_STRUCT
{

    ULONG           ux_transfer_request_status;
    struct UX_ENDPOINT_STRUCT
                    *ux_transfer_request_endpoint;
    UCHAR *         ux_transfer_request_data_pointer;
    ULONG           ux_transfer_request_requested_length;
    ULONG           ux_transfer_request_actual_length;
    UINT            ux_transfer_request_type;
    UINT            ux_transfer_request_function;
    UINT            ux_transfer_request_value;
    UINT            ux_transfer_request_index;
    VOID            (*ux_transfer_request_completion_function)
                                  (struct UX_TRANSFER_STRUCT *);
    TX_SEMAPHORE    ux_transfer_request_semaphore;
    VOID            *ux_transfer_request_class_instance;
```

```
    ULONG           ux_transfer_request_maximum_length;
    UINT            ux_transfer_request_completion_code;
    ULONG           ux_transfer_request_packet_length;
    struct UX_TRANSFER_STRUCT
                    *ux_transfer_request_next_transfer_request;
    VOID            *ux_transfer_request_user_specific;
} UX_TRANSFER;


typedef struct UX_ENDPOINT_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bEndpointAddress;
    ULONG           bmAttributes;
    ULONG           wMaxPacketSize;
    ULONG           bInterval;
} UX_ENDPOINT_DESCRIPTOR;


typedef struct UX_ENDPOINT_STRUCT
{

    ULONG           ux_endpoint;
    ULONG           ux_endpoint_state;
    void            *ux_endpoint_ed;
    struct UX_ENDPOINT_DESCRIPTOR_STRUCT
                    ux_endpoint_descriptor;
    struct UX_ENDPOINT_STRUCT
                    *ux_endpoint_next_endpoint;
    struct UX_INTERFACE_STRUCT
                    *ux_endpoint_interface;
    struct UX_DEVICE_STRUCT
                    *ux_endpoint_device;
    struct UX_TRANSFER_STRUCT
                    ux_endpoint_transfer_request;
} UX_ENDPOINT;



typedef struct UX_DEVICE_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bcdUSB;
    ULONG           bDeviceClass;
    ULONG           bDeviceSubClass;
    ULONG           bDeviceProtocol;
    ULONG           bMaxPacketSize0;
    ULONG           idVendor;
    ULONG           idProduct;
    ULONG           bcdDevice;
    ULONG           iManufacturer;
    ULONG           iProduct;
    ULONG           iSerialNumber;
    ULONG           bNumConfigurations;
} UX_DEVICE_DESCRIPTOR;


typedef struct UX_DEVICE_QUALIFIER_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bcdUSB;
    ULONG           bDeviceClass;
    ULONG           bDeviceSubClass;
    ULONG           bDeviceProtocol;
```

```
    ULONG           bMaxPacketSize0;
    ULONG           bNumConfigurations;
    ULONG           bReserved;
} UX_DEVICE_QUALIFIER_DESCRIPTOR;


typedef struct UX_OTHER_SPEED_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           wTotalLength;
    ULONG           bNumInterfaces;
    ULONG           bConfigurationValue;
    ULONG           iConfiguration;
    ULONG           bmAttributes;
    ULONG           MaxPower;
} UX_OTHER_SPEED_DESCRIPTOR;

typedef struct UX_DEVICE_STRUCT
{

    ULONG           ux_device_handle;
    ULONG           ux_device_type;
    ULONG           ux_device_state;
    ULONG           ux_device_address;
    ULONG           ux_device_speed;
    ULONG           ux_device_port_location;
    ULONG           ux_device_max_power;
    ULONG           ux_device_power_source;
    UINT            ux_device_current_configuration;
    struct UX_DEVICE_STRUCT
                    *ux_device_parent;
    struct UX_HOST_CLASS_STRUCT
                    *ux_device_class;
    VOID            *ux_device_class_instance;
    struct UX_HCD_STRUCT
                    *ux_device_hcd;
    struct UX_CONFIGURATION_STRUCT
                    *ux_device_first_configuration;
    struct UX_DEVICE_STRUCT
                    *ux_device_next_device;
    struct UX_DEVICE_DESCRIPTOR_STRUCT
                    ux_device_descriptor;
    struct UX_ENDPOINT_STRUCT
                    ux_device_control_endpoint;
    struct UX_HUB_TT_STRUCT
                    ux_device_hub_tt[UX_MAX_TT];
} UX_DEVICE;


typedef struct UX_CONFIGURATION_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           wTotalLength;
    ULONG           bNumInterfaces;
    ULONG           bConfigurationValue;
    ULONG           iConfiguration;
    ULONG           bmAttributes;
    ULONG           MaxPower;
} UX_CONFIGURATION_DESCRIPTOR;


typedef struct UX_CONFIGURATION_STRUCT
{

    ULONG           ux_configuration_handle;
    ULONG           ux_configuration_state;
```

```
    struct UX_CONFIGURATION_DESCRIPTOR_STRUCT
                    ux_configuration_descriptor;
    struct UX_INTERFACE_STRUCT
                    *ux_configuration_first_interface;
    struct UX_CONFIGURATION_STRUCT
                    *ux_configuration_next_configuration;
    struct UX_DEVICE_STRUCT
                    *ux_configuration_device;
} UX_CONFIGURATION;


typedef struct UX_INTERFACE_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bInterfaceNumber;
    ULONG           bAlternateSetting;
    ULONG           bNumEndpoints;
    ULONG           bInterfaceClass;
    ULONG           bInterfaceSubClass;
    ULONG           bInterfaceProtocol;
    ULONG           iInterface;
} UX_INTERFACE_DESCRIPTOR;

typedef struct UX_INTERFACE_STRUCT
{

    ULONG           ux_interface_handle;
    ULONG           ux_interface_state;
    UINT            ux_interface_current_alternate_setting;
    struct UX_INTERFACE_DESCRIPTOR_STRUCT
                    ux_interface_descriptor;
    struct UX_HOST_CLASS_STRUCT
                    *ux_interface_class;
    VOID            *ux_interface_class_instance;
    struct UX_ENDPOINT_STRUCT
                    *ux_interface_first_endpoint;
    struct UX_INTERFACE_STRUCT
                    *ux_interface_next_interface;
    struct UX_CONFIGURATION_STRUCT
                    *ux_interface_configuration;
} UX_INTERFACE;


typedef struct UX_STRING_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bString[1];
} UX_STRING_DESCRIPTOR;

typedef struct UX_HCD_STRUCT
{

    UCHAR           ux_hcd_name[32];
    UCHAR           ux_hcd_address[16];
    UINT            ux_hcd_status;
    UINT            ux_hcd_controller_type;
    UINT            ux_hcd_irq;
    UINT            ux_hcd_nb_root_hubs;
    UINT            ux_hcd_root_hub_signal;
    UINT            ux_hcd_nb_devices;
    UINT            ux_hcd_power_switch;
    UINT            ux_hcd_thread_signal;
    ULONG           ux_hcd_rh_device_connection;
    ULONG           ux_hcd_io;
    ULONG           ux_hcd_available_bandwidth;
    ULONG           ux_hcd_maximum_transfer_request_size;
```

```
    ULONG           ux_hcd_version;
    UINT            (*ux_hcd_entry_function) (struct UX_HCD_STRUCT *, UINT, VOID *);
    void            *ux_hcd_controller_hardware;
} UX_HCD;
```

USBX Device Stack Data Types

```
typedef struct UX_SLAVE_TRANSFER_STRUCT
{

    ULONG           ux_slave_transfer_request_status;
    ULONG           ux_slave_transfer_request_type;
    struct UX_SLAVE_ENDPOINT_STRUCT
                    *ux_slave_transfer_request_endpoint;
    UCHAR *         ux_slave_transfer_request_data_pointer;
    UCHAR *         ux_slave_transfer_request_current_data_pointer;
    ULONG           ux_slave_transfer_request_requested_length;
    ULONG           ux_slave_transfer_request_actual_length;
    ULONG           ux_slave_transfer_request_in_transfer_length;
    ULONG           ux_slave_transfer_request_completion_code;
    ULONG           ux_slave_transfer_request_phase;
    VOID            (*ux_slave_transfer_request_completion_function)
                                    (struct UX_SLAVE_TRANSFER_STRUCT *);
    TX_SEMAPHORE    ux_slave_transfer_request_semaphore;
    ULONG           ux_slave_transfer_request_timeout;
    ULONG           ux_slave_transfer_request_force_zlp;
    UCHAR           ux_slave_transfer_request_setup[UX_SETUP_SIZE];
    ULONG           ux_slave_transfer_request_status_phase_ignore;
} UX_SLAVE_TRANSFER;


typedef struct UX_SLAVE_ENDPOINT_STRUCT
{

    ULONG           ux_slave_endpoint_status;
    ULONG           ux_slave_endpoint_state;
    void            *ux_slave_endpoint_ed;
    struct UX_ENDPOINT_DESCRIPTOR_STRUCT
                    ux_slave_endpoint_descriptor;
    struct UX_SLAVE_ENDPOINT_STRUCT
                    *ux_slave_endpoint_next_endpoint;
    struct UX_SLAVE_INTERFACE_STRUCT
                    *ux_slave_endpoint_interface;
    struct UX_SLAVE_DEVICE_STRUCT
                    *ux_slave_endpoint_device;
    struct UX_SLAVE_TRANSFER_STRUCT
                    ux_slave_endpoint_transfer_request;
} UX_SLAVE_ENDPOINT;


typedef struct UX_SLAVE_INTERFACE_STRUCT
{
    ULONG           ux_slave_interface_status;
    struct UX_SLAVE_CLASS_STRUCT
                    *ux_slave_interface_class;
    VOID            *ux_slave_interface_class_instance;

    struct UX_INTERFACE_DESCRIPTOR_STRUCT
                    ux_slave_interface_descriptor;
    struct UX_SLAVE_INTERFACE_STRUCT
                    *ux_slave_interface_next_interface;
    struct UX_SLAVE_ENDPOINT_STRUCT
```

```
                    *ux_slave_interface_first_endpoint;
} UX_SLAVE_INTERFACE;


typedef struct UX_SLAVE_DEVICE_STRUCT
{

    ULONG           ux_slave_device_state;
    struct UX_DEVICE_DESCRIPTOR_STRUCT
                    ux_slave_device_descriptor;
    struct UX_SLAVE_ENDPOINT_STRUCT
                    ux_slave_device_control_endpoint;
    ULONG           ux_slave_device_configuration_selected;
    struct UX_CONFIGURATION_DESCRIPTOR_STRUCT
                    ux_slave_device_configuration_descriptor;
    struct UX_SLAVE_INTERFACE_STRUCT
                    *ux_slave_device_first_interface;
    struct UX_SLAVE_INTERFACE_STRUCT
                                        *ux_slave_device_interfaces_pool;
    ULONG                               ux_slave_device_interfaces_pool_number;
    struct UX_SLAVE_ENDPOINT_STRUCT
                                        *ux_slave_device_endpoints_pool;
    ULONG                               ux_slave_device_endpoints_pool_number;

} UX_SLAVE_DEVICE;


typedef struct UX_SLAVE_DCD_STRUCT
{

    UCHAR           ux_slave_dcd_name[32];
    UINT            ux_slave_dcd_status;
    UINT            ux_slave_dcd_controller_type;
    UINT            ux_slave_dcd_irq;
    ULONG           ux_slave_dcd_io;
    ULONG           ux_slave_dcd_device_address;
    UINT            (*ux_slave_dcd_function)
                                (struct UX_SLAVE_DCD_STRUCT *,UINT, VOID *);
    void            *ux_slave_dcd_controller_hardware;
    struct UX_SLAVE_DEVICE_STRUCT
                    ux_slave_dcd_device;
} UX_SLAVE_DCD;

typedef struct UX_SLAVE_CLASS_COMMAND_STRUCT
{

    UINT            ux_slave_class_command_request;
    VOID            *ux_slave_class_command_container;
    VOID            *ux_slave_class_command_instance;
    UINT            ux_slave_class_command_pid;
    UINT            ux_slave_class_command_vid;
    UINT            ux_slave_class_command_class;
    UINT            ux_slave_class_command_subclass;
    UINT            ux_slave_class_command_protocol;
    struct UX_SLAVE_CLASS_STRUCT
                    *ux_slave_class_command_class_ptr;
} UX_SLAVE_CLASS_COMMAND;


typedef struct UX_SLAVE_CLASS_STRUCT
{

    UCHAR           ux_slave_class_name[32];
    UINT            ux_slave_class_status;
    UINT            (*ux_slave_class_entry_function)
                                (struct UX_SLAVE_CLASS_COMMAND_STRUCT *) ;
    VOID            *ux_slave_class_instance;
    VOID            *ux_slave_class_client;
    TX_THREAD       ux_slave_class_thread;
    VOID            *ux_slave_class_thread_stack;
```

```
    VOID            *ux_slave_class_interface_parameter;
    ULONG           ux_slave_class_interface_number;
    struct UX_SLAVE_INTERFACE_STRUCT
                    *ux_slave_class_interface;

} UX_SLAVE_CLASS;


USBX SYSTEM DATA TYPES

typedef struct UX_MEMORY_BLOCK_STRUCT
{

    ULONG           ux_memory_block_size;
    ULONG           ux_memory_block_status;
    struct  UX_MEMORY_BLOCK_STRUCT
                    *ux_memory_block_next;
    struct  UX_MEMORY_BLOCK_STRUCT
                    *ux_memory_block_previous;
} UX_MEMORY_BLOCK;


typedef struct UX_SYSTEM_STRUCT
{

    UX_MEMORY_BLOCK *ux_system_regular_memory_pool_start;
    ULONG           ux_system_regular_memory_pool_size;
    UX_MEMORY_BLOCK *ux_system_cache_safe_memory_pool_start;
    ULONG           ux_system_cache_safe_memory_pool_size;
    TX_MUTEX        ux_system_mutex;
} UX_SYSTEM;

typedef struct UX_SYSTEM_HOST_STRUCT
{

    UINT            ux_system_host_max_class;
    UINT            ux_system_host_registered_class;
    UX_HOST_CLASS   *ux_system_host_class_array;
    UINT            ux_system_host_max_hcd;
    UX_HCD          *ux_system_host_hcd_array;
    UINT            ux_system_host_registered_hcd;
    UX_DEVICE       *ux_system_host_device_array;
    ULONG           ux_system_host_max_devices;
    ULONG           ux_system_host_max_ed;
    ULONG           ux_system_host_max_td;
    ULONG           ux_system_host_max_iso_td;
    UINT            ux_system_host_rhsc_hcd;
    UCHAR *         ux_system_host_enum_thread_stack;
    TX_THREAD       ux_system_host_enum_thread;
    TX_SEMAPHORE    ux_system_host_enum_semaphore;
    VOID            (*ux_system_host_enum_hub_function) (VOID);
    UCHAR *         ux_system_host_hcd_thread_stack;
    TX_THREAD       ux_system_host_hcd_thread;
    TX_SEMAPHORE    ux_system_host_hcd_semaphore;
    UINT            (*ux_system_host_change_function) (ULONG, UX_HOST_CLASS *);
} UX_SYSTEM_HOST;


typedef struct UX_SYSTEM_SLAVE_STRUCT
{

    TX_THREAD       ux_system_slave_enum_thread;
    UCHAR *         ux_system_slave_enum_thread_stack;
    TX_SEMAPHORE    ux_system_slave_enum_semaphore;
    UX_SLAVE_DCD    ux_system_slave_dcd;
    UX_SLAVE_DEVICE ux_system_slave_device;
    UCHAR *         ux_system_slave_device_framework;
    ULONG           ux_system_slave_device_framework_length;
    UCHAR *         ux_system_slave_device_framework_full_speed;
    ULONG           ux_system_slave_device_framework_length_full_speed;
```

```
    UCHAR *         ux_system_slave_device_framework_high_speed;
    ULONG           ux_system_slave_device_framework_length_high_speed;
    UCHAR *         ux_system_slave_string_framework;
    ULONG           ux_system_slave_string_framework_length;
    UCHAR *         ux_system_slave_language_id_framework;
    ULONG           ux_system_slave_language_id_framework_length;
    UINT            ux_system_slave_max_class;
    UINT            ux_system_slave_registered_class;
    UX_SLAVE_CLASS  *ux_system_slave_class_array;
    ULONG           ux_system_slave_speed;
    ULONG           ux_system_slave_power_state;
    ULONG           ux_system_slave_remote_wakeup_capability;
    UINT            (*ux_system_slave_change_function) (ULONG);

} UX_SYSTEM_SLAVE;
```

USBX Classes Data Types

```
typedef struct UX_HOST_CLASS_AUDIO_INTERFACE_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bDescriptorSubType;
    ULONG           bFormatType;
    ULONG           bNrChannels;
    ULONG           bSubframeSize;
    ULONG           bBitResolution;
    ULONG           bSamFreqType;
} UX_HOST_CLASS_AUDIO_INTERFACE_DESCRIPTOR;


typedef struct UX_HOST_CLASS_AUDIO_INPUT_TERMINAL_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bDescriptorSubType;
    ULONG           bTerminalID;
    ULONG           wTerminalType;
    ULONG           bAssocTerminal;
    ULONG           bNrChannels;
    ULONG           wChannelConfig;
    ULONG           iChannelNames;
    ULONG           iTerminal;
} UX_HOST_CLASS_AUDIO_INPUT_TERMINAL_DESCRIPTOR;


typedef struct UX_HOST_CLASS_AUDIO_OUTPUT_TERMINAL_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bDescriptorSubType;
    ULONG           bTerminalID;
    ULONG           wTerminalType;
    ULONG           bAssocTerminal;
    ULONG           bSourceID;
    ULONG           iTerminal;
} UX_HOST_CLASS_AUDIO_OUTPUT_TERMINAL_DESCRIPTOR;
```

```
typedef struct UX_HOST_CLASS_AUDIO_FEATURE_UNIT_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bDescriptorSubType;
    ULONG           bUnitID;
    ULONG           bSourceID;
    ULONG           bControlSize;
    ULONG           bmaControls;
} UX_HOST_CLASS_AUDIO_FEATURE_UNIT_DESCRIPTOR;


typedef struct UX_HOST_CLASS_AUDIO_STREAMING_INTERFACE_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bDescriptorSubtype;
    ULONG           bTerminalLink;
    ULONG           bDelay;
    ULONG           wFormatTag;
} UX_HOST_CLASS_AUDIO_STREAMING_INTERFACE_DESCRIPTOR;


typedef struct UX_HOST_CLASS_AUDIO_STREAMING_ENDPOINT_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bDescriptorSubtype;
    ULONG           bmAttributes;
    ULONG           bLockDelayUnits;
    ULONG           wLockDelay;
} UX_HOST_CLASS_AUDIO_STREAMING_ENDPOINT_DESCRIPTOR;


typedef struct UX_HOST_CLASS_AUDIO_STRUCT
{

    struct UX_HOST_CLASS_AUDIO_STRUCT
                    *ux_host_class_audio_next_instance;
    UX_HOST_CLASS   *ux_host_class_audio_class;
    UX_DEVICE       *ux_host_class_audio_device;
    UX_INTERFACE    *ux_host_class_audio_streaming_interface;
    ULONG           ux_host_class_audio_control_interface_number;
    UX_ENDPOINT     *ux_host_class_audio_isochronous_endpoint;
    struct UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST_STRUCT
                    *ux_host_class_audio_head_transfer_request;
    struct UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST_STRUCT
                    *ux_host_class_audio_tail_transfer_request;
    UINT            ux_host_class_audio_state;
    ULONG           ux_host_class_audio_terminal_link;
    ULONG           ux_host_class_audio_type;
    UCHAR *         ux_host_class_audio_configuration_descriptor;
    ULONG           ux_host_class_audio_configuration_descriptor_length;
    ULONG           ux_host_class_audio_feature_unit_id;
    UINT            ux_host_class_audio_channels;
    ULONG           ux_host_class_audio_channel_control[UX_HOST_CLASS_AUDIO_MAX_CHANNEL];
    UCHAR           ux_host_class_audio_name[UX_HOST_CLASS_AUDIO_NAME_LENGTH];
    TX_SEMAPHORE    ux_host_class_audio_semaphore;
} UX_HOST_CLASS_AUDIO;


typedef struct UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST_STRUCT
{

    ULONG           ux_host_class_audio_transfer_request_status;
    UCHAR *         ux_host_class_audio_transfer_request_data_pointer;
    ULONG           ux_host_class_audio_transfer_request_requested_length;
```

```
    ULONG           ux_host_class_audio_transfer_request_actual_length;
    VOID            (*ux_host_class_audio_transfer_request_completion_function)
                            (struct UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST_STRUCT *);
    TX_SEMAPHORE    ux_host_class_audio_transfer_request_semaphore;
    VOID            *ux_host_class_audio_transfer_request_class_instance;
    UINT            ux_host_class_audio_transfer_request_completion_code;
    struct UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST_STRUCT
                    *ux_host_class_audio_transfer_request_next_audio_transfer_request;
    UX_TRANSFER     ux_host_class_audio_transfer_request;
} UX_HOST_CLASS_AUDIO_TRANSFER_REQUEST;


typedef struct UX_HOST_CLASS_AUDIO_CONTROL_STRUCT
{

    ULONG           ux_host_class_audio_control;
    ULONG           ux_host_class_audio_control_channel;
    ULONG           ux_host_class_audio_control_min;
    ULONG           ux_host_class_audio_control_max;
    ULONG           ux_host_class_audio_control_res;
    ULONG           ux_host_class_audio_control_cur;
} UX_HOST_CLASS_AUDIO_CONTROL;


typedef struct UX_HOST_CLASS_AUDIO_CHANNEL_STRUCT
{

    ULONG           ux_host_class_audio_channel_control;
    ULONG           ux_host_class_audio_channel;
} UX_HOST_CLASS_AUDIO_CHANNEL;


typedef struct UX_HOST_CLASS_AUDIO_SAMPLING_STRUCT
{

    ULONG           ux_host_class_audio_sampling_channels;
    ULONG           ux_host_class_audio_sampling_frequency;
    ULONG           ux_host_class_audio_sampling_resolution;
} UX_HOST_CLASS_AUDIO_SAMPLING;


typedef struct UX_HOST_CLASS_AUDIO_SAMPLING_CHARACTERISTICS_STRUCT
{

    ULONG           ux_host_class_audio_sampling_characteristics_channels;
    ULONG           ux_host_class_audio_sampling_characteristics_frequency_low;
    ULONG           ux_host_class_audio_sampling_characteristics_frequency_high;
    ULONG           ux_host_class_audio_sampling_characteristics_resolution;
} UX_HOST_CLASS_AUDIO_SAMPLING_CHARACTERISTICS;



typedef struct UX_HID_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bcdHID;
    ULONG           bCountryCode;
    ULONG           bNumDescriptor;
    ULONG           bReportDescriptorType;
    ULONG           wItemLength;
} UX_HID_DESCRIPTOR;


typedef struct UX_HOST_CLASS_HID_REPORT_CALLBACK_STRUCT
{

    struct UX_HOST_CLASS_HID_CLIENT_STRUCT
                    *ux_host_class_hid_report_callback_client;
```

```
    ULONG           ux_host_class_hid_report_callback_id;
    ULONG           ux_host_class_hid_report_callback_status;
    ULONG           ux_host_class_hid_report_callback_flags;
    ULONG           ux_host_class_hid_report_callback_value;
    ULONG           ux_host_class_hid_report_callback_usage;
    ULONG           ux_host_class_hid_report_callback_length;
    ULONG           ux_host_class_hid_report_callback_actual_length;
    VOID            *ux_host_class_hid_report_callback_buffer;
    VOID            (*ux_host_class_hid_report_callback_function)
                            (struct UX_HOST_CLASS_HID_REPORT_CALLBACK_STRUCT *);
} UX_HOST_CLASS_HID_REPORT_CALLBACK;


typedef struct UX_HOST_CLASS_HID_REPORT_GET_ID_STRUCT
{

    ULONG           ux_host_class_hid_report_get_id;
    ULONG           ux_host_class_hid_report_get_type;
    struct UX_HOST_CLASS_HID_REPORT_STRUCT
                    *ux_host_class_hid_report_get_report;
} UX_HOST_CLASS_HID_REPORT_GET_ID;


typedef struct UX_HOST_CLASS_HID_LOCAL_ITEM_STRUCT
{

    ULONG           ux_host_class_hid_local_item_usages[UX_HOST_CLASS_HID_USAGES];
    ULONG           ux_host_class_hid_local_item_number_usage;
    SLONG           ux_host_class_hid_local_item_usage_min;
    SLONG           ux_host_class_hid_local_item_usage_max;
    ULONG           ux_host_class_hid_local_item_delimiter_level;
    ULONG           ux_host_class_hid_local_item_delimiter_branch;
} UX_HOST_CLASS_HID_LOCAL_ITEM;


typedef struct UX_HOST_CLASS_HID_GLOBAL_ITEM_STRUCT
{

    ULONG           ux_host_class_hid_global_item_usage_page;
    SLONG           ux_host_class_hid_global_item_logical_min;
    SLONG           ux_host_class_hid_global_item_logical_max;
    SLONG           ux_host_class_hid_global_item_physical_min;
    SLONG           ux_host_class_hid_global_item_physical_max;
    ULONG           ux_host_class_hid_global_item_unit_expo;
    ULONG           ux_host_class_hid_global_item_unit;
    ULONG           ux_host_class_hid_global_item_report_size;
    ULONG           ux_host_class_hid_global_item_report_id;
    ULONG           ux_host_class_hid_global_item_report_count;
} UX_HOST_CLASS_HID_GLOBAL_ITEM;


typedef struct UX_HOST_CLASS_HID_FIELD_STRUCT
{

    ULONG           ux_host_class_hid_field_physical;
    ULONG           ux_host_class_hid_field_logical;
    ULONG           ux_host_class_hid_field_application;
    SLONG           ux_host_class_hid_field_logical_min;
    SLONG           ux_host_class_hid_field_logical_max;
    SLONG           ux_host_class_hid_field_physical_min;
    SLONG           ux_host_class_hid_field_physical_max;
    ULONG           ux_host_class_hid_field_unit;
    ULONG           ux_host_class_hid_field_unit_expo;
    ULONG           ux_host_class_hid_field_report_type;
    ULONG           ux_host_class_hid_field_report_id;
    ULONG           ux_host_class_hid_field_report_offset;
    ULONG           ux_host_class_hid_field_report_size;
    ULONG           ux_host_class_hid_field_report_count;
    ULONG           ux_host_class_hid_field_value;
```

```
    ULONG           *ux_host_class_hid_field_usages;
    ULONG           ux_host_class_hid_field_number_usage;
    ULONG           *ux_host_class_hid_field_values;
    ULONG           ux_host_class_hid_field_number_values;
    struct UX_HOST_CLASS_HID_REPORT
                    *ux_host_class_hid_field_report;
    struct UX_HOST_CLASS_HID_FIELD_STRUCT
                    *ux_host_class_hid_field_next_field;
} UX_HOST_CLASS_HID_FIELD;


typedef struct UX_HOST_CLASS_HID_REPORT_STRUCT
{

    ULONG           ux_host_class_hid_report_id;
    ULONG           ux_host_class_hid_report_type;
    struct UX_HOST_CLASS_HID_FIELD_STRUCT
                    *ux_host_class_hid_report_field;
    ULONG           ux_host_class_hid_report_number_field;
    ULONG           ux_host_class_hid_report_byte_length;
    ULONG           ux_host_class_hid_report_bit_length;
    ULONG           ux_host_class_hid_report_callback_flags;
    VOID            *ux_host_class_hid_report_callback_buffer;
    ULONG           ux_host_class_hid_report_callback_length;
    VOID            (*ux_host_class_hid_report_callback_function) (struct
UX_HOST_CLASS_HID_REPORT_CALLBACK_STRUCT *);
    struct UX_HOST_CLASS_HID_REPORT_STRUCT
                    *ux_host_class_hid_report_next_report;
} UX_HOST_CLASS_HID_REPORT;


typedef struct UX_HOST_CLASS_HID_PARSER_STRUCT
{

    UX_HOST_CLASS_HID_GLOBAL_ITEM
                    ux_host_class_hid_parser_global;
    UX_HOST_CLASS_HID_GLOBAL_ITEM
                    ux_host_class_hid_parser_global_pool[UX_HOST_CLASS_HID_MAX_GLOBAL];
    ULONG           ux_host_class_hid_parser_number_global;
    UX_HOST_CLASS_HID_LOCAL_ITEM
                    ux_host_class_hid_parser_local;
    ULONG           ux_host_class_hid_parser_application;
    ULONG           ux_host_class_hid_parser_collection
                                    [UX_HOST_CLASS_HID_MAX_COLLECTION];
    ULONG           ux_host_class_hid_parser_number_collection;
    ULONG           ux_host_class_hid_parser_main_page;
    ULONG           ux_host_class_hid_parser_main_usage;
    UX_HOST_CLASS_HID_REPORT
                    *ux_host_class_hid_parser_input_report;
    UX_HOST_CLASS_HID_REPORT
                    *ux_host_class_hid_parser_output_report;
    UX_HOST_CLASS_HID_REPORT
                    *ux_host_class_hid_parser_feature_report;
} UX_HOST_CLASS_HID_PARSER;


typedef struct UX_HOST_CLASS_HID_ITEM_STRUCT
{

    UCHAR           ux_host_class_hid_item_report_type;
    UCHAR           ux_host_class_hid_item_report_tag;
    USHORT          ux_host_class_hid_item_report_length;
    USHORT          ux_host_class_hid_item_report_format;
}  UX_HOST_CLASS_HID_ITEM;


typedef struct UX_HOST_CLASS_HID_STRUCT
{

    struct UX_HOST_CLASS_HID_STRUCT
```

```
                    *ux_host_class_hid_next_instance;
    UX_HOST_CLASS   *ux_host_class_hid_class;
    UX_DEVICE       *ux_host_class_hid_device;
    UX_ENDPOINT     *ux_host_class_hid_interrupt_endpoint;
    UINT            ux_host_class_hid_interrupt_endpoint_status;
    UX_INTERFACE    *ux_host_class_hid_interface;
    ULONG           ux_host_class_hid_state;
    struct UX_HID_DESCRIPTOR_STRUCT
                    ux_host_class_hid_descriptor;
    UX_HOST_CLASS_HID_PARSER
                    ux_host_class_hid_parser;
    struct UX_HOST_CLASS_HID_CLIENT_STRUCT
                    *ux_host_class_hid_client;
    TX_SEMAPHORE    ux_host_class_hid_semaphore;
} UX_HOST_CLASS_HID;


typedef struct UX_HOST_CLASS_HID_CLIENT_COMMAND_STRUCT
{

    UINT            ux_host_class_hid_client_command_request;
    VOID            *ux_host_class_hid_client_command_container;
    UX_HOST_CLASS_HID    *ux_host_class_hid_client_command_instance;
    ULONG           ux_host_class_hid_client_command_page;
    ULONG           ux_host_class_hid_client_command_usage;
} UX_HOST_CLASS_HID_CLIENT_COMMAND;


typedef struct UX_HOST_CLASS_HID_CLIENT_REPORT_STRUCT
{

    UX_HOST_CLASS_HID_REPORT
                    *ux_host_class_hid_client_report;
    ULONG           *ux_host_class_hid_client_report_buffer;
    ULONG           ux_host_class_hid_client_report_length;
    ULONG           ux_host_class_hid_client_report_actual_length;
    UINT            ux_host_class_hid_client_report_flags;
} UX_HOST_CLASS_HID_CLIENT_REPORT;


typedef struct UX_HOST_CLASS_HID_CLIENT_STRUCT
{

    ULONG           ux_host_class_hid_client_status;
    UCHAR           ux_host_class_hid_client_name[64];
    UINT            (*ux_host_class_hid_client_handler)
                            (struct UX_HOST_CLASS_HID_CLIENT_COMMAND_STRUCT *);
    VOID            *ux_host_class_hid_client_local_instance;
} UX_HOST_CLASS_HID_CLIENT;


typedef struct UX_HOST_CLASS_HID_MOUSE_STRUCT
{

    ULONG           ux_host_class_hid_mouse_state;
    UX_HOST_CLASS_HID   *ux_host_class_hid_mouse_hid;
    USHORT          ux_host_class_hid_mouse_id;
    SLONG           ux_host_class_hid_mouse_x_position;
    SLONG           ux_host_class_hid_mouse_y_position;
    ULONG           ux_host_class_hid_mouse_buttons;
} UX_HOST_CLASS_HID_MOUSE;


typedef struct UX_HOST_CLASS_HID_KEYBOARD_STRUCT
{

    ULONG           ux_host_class_hid_keyboard_state;
    UX_HOST_CLASS_HID   *ux_host_class_hid_keyboard_hid;
    USHORT          ux_host_class_hid_keyboard_id;
    TX_THREAD       ux_host_class_hid_keyboard_thread;
```

```
    TX_SEMAPHORE    ux_host_class_hid_keyboard_semaphore;
    ULONG           ux_host_class_hid_keyboard_alternate_key_state;
    ULONG           ux_host_class_hid_keyboard_led_mask;
    VOID            *ux_host_class_hid_keyboard_thread_stack;
    ULONG           *ux_host_class_hid_keyboard_usage_array;
    ULONG           *ux_host_class_hid_keyboard_usage_array_head;
    ULONG           *ux_host_class_hid_keyboard_usage_array_tail;
} UX_HOST_CLASS_HID_KEYBOARD;


typedef struct UX_HOST_CLASS_HID_REMOTE_CONTROL_STRUCT
{

    ULONG           ux_host_class_hid_remote_control_state;
    ULONG           *ux_host_class_hid_remote_control_usage_array;
    ULONG           *ux_host_class_hid_remote_control_usage_array_head;
    ULONG           *ux_host_class_hid_remote_control_usage_array_tail;
} UX_HOST_CLASS_HID_REMOTE_CONTROL;


typedef struct UX_HUB_DESCRIPTOR_STRUCT
{

    ULONG           bLength;
    ULONG           bDescriptorType;
    ULONG           bNbPorts;
    ULONG           wHubCharacteristics;
    ULONG           bPwrOn2PwrGood;
    ULONG           bHubContrCurrent;
    ULONG           bDeviceRemovable;
    ULONG           bPortPwrCtrlMask;
} UX_HUB_DESCRIPTOR;


typedef struct UX_HOST_CLASS_HUB_STRUCT
{

    struct UX_HOST_CLASS_HUB_STRUCT
                    *ux_host_class_hub_next_instance;
    UX_HOST_CLASS   *ux_host_class_hub_class;
    UX_DEVICE       *ux_host_class_hub_device;
    UX_ENDPOINT     *ux_host_class_hub_interrupt_endpoint;
    UX_INTERFACE    *ux_host_class_hub_interface;
    UINT            ux_host_class_hub_instance_status;
    UINT            ux_host_class_hub_state;
    UINT            ux_host_class_hub_enumeration_retry_count;
    UINT            ux_host_class_hub_change_semaphore;
    struct UX_HUB_DESCRIPTOR_STRUCT
                    ux_host_class_hub_descriptor;
    TX_SEMAPHORE    ux_host_class_hub_semaphore;
} UX_HOST_CLASS_HUB;


typedef struct UX_HOST_CLASS_PRINTER_STRUCT
{

    struct UX_HOST_CLASS_PRINTER_STRUCT
                    *ux_host_class_printer_next_instance;
    UX_HOST_CLASS   *ux_host_class_printer_class;
    UX_DEVICE       *ux_host_class_printer_device;
    UX_INTERFACE    *ux_host_class_printer_interface;
    UX_ENDPOINT     *ux_host_class_printer_bulk_out_endpoint;
    UX_ENDPOINT     *ux_host_class_printer_bulk_in_endpoint;
    UINT            ux_host_class_printer_state;
    UCHAR           ux_host_class_printer_name[UX_HOST_CLASS_PRINTER_NAME_LENGTH];
    TX_SEMAPHORE    ux_host_class_printer_semaphore;
} UX_HOST_CLASS_PRINTER;
```

```
typedef struct UX_HOST_CLASS_STORAGE_MEDIA_STRUCT
{

    FX_MEDIA                                ux_host_class_storage_media;
    ULONG                                   ux_host_class_storage_media_status;
    ULONG                                   ux_host_class_storage_media_lun;
    ULONG                                   ux_host_class_storage_media_partition_start;
    VOID                                    *ux_host_class_storage_media_memory;

} UX_HOST_CLASS_STORAGE_MEDIA;


typedef struct UX_HOST_CLASS_STORAGE_STRUCT
{

    struct UX_HOST_CLASS_STORAGE_STRUCT
                    *ux_host_class_storage_next_instance;
    UX_HOST_CLASS   *ux_host_class_storage_class;
    UX_DEVICE       *ux_host_class_storage_device;
    UX_INTERFACE    *ux_host_class_storage_interface;
    UX_ENDPOINT     *ux_host_class_storage_bulk_out_endpoint;
    UX_ENDPOINT     *ux_host_class_storage_bulk_in_endpoint;
    UX_ENDPOINT     *ux_host_class_storage_interrupt_endpoint;
    UINT            ux_host_class_storage_state;
    UINT            ux_host_class_storage_media_type;
    UINT            ux_host_class_storage_removable_media;
    UINT            ux_host_class_storage_write_protected_media;
    UINT            ux_host_class_storage_max_lun;
    UINT            ux_host_class_storage_lun;
    ULONG           ux_host_class_storage_sector_size;
    UINT            (*ux_host_class_storage_transport)
                                (struct UX_HOST_CLASS_STORAGE_STRUCT *storage,
                                        UCHAR * data_pointer);
    UCHAR           ux_host_class_storage_cbw[UX_HOST_CLASS_STORAGE_CBW_LENGTH];
    UCHAR           ux_host_class_storage_saved_cbw[UX_HOST_CLASS_STORAGE_CBW_LENGTH];
    UCHAR           ux_host_class_storage_csw[UX_HOST_CLASS_STORAGE_CSW_LENGTH];
    ULONG           ux_host_class_storage_sense_code;
    UCHAR *         ux_host_class_storage_memory;
    TX_SEMAPHORE    ux_host_class_storage_semaphore;
} UX_HOST_CLASS_STORAGE;


typedef struct UX_SLAVE_CLASS_STORAGE_LUN_STRUCT
{
    ULONG           ux_slave_class_storage_media_last_lba;
    ULONG           ux_slave_class_storage_media_block_length;
    ULONG           ux_slave_class_storage_media_type;
    ULONG           ux_slave_class_storage_media_removable_flag;
    ULONG           ux_slave_class_storage_media_id;
    ULONG           ux_slave_class_storage_scsi_tag;
    UCHAR           ux_slave_class_storage_request_sense_key;
    UCHAR           ux_slave_class_storage_request_code;
    UCHAR           ux_slave_class_storage_request_code_qualifier;
    UINT            (*ux_slave_class_storage_media_read)
                         (VOID *storage, ULONG lun, UCHAR * data_pointer,
                          ULONG number_blocks, ULONG lba);
    UINT            (*ux_slave_class_storage_media_write)(VOID *storage,
                             ULONG lun, UCHAR * data_pointer, ULONG number_blocks,
                             ULONG lba);
    UINT            (*ux_slave_class_storage_media_status)(VOID *storage,
                             ULONG lun, ULONG media_id);
} UX_SLAVE_CLASS_STORAGE_LUN;


typedef struct UX_SLAVE_CLASS_STORAGE_STRUCT
{
    UX_SLAVE_INTERFACE          *ux_slave_class_storage_interface;
    ULONG                       ux_slave_class_storage_number_lun;
```

```
    UX_SLAVE_CLASS_STORAGE_LUN  ux_slave_class_storage_lun[UX_MAX_SLAVE_LUN];

} UX_SLAVE_CLASS_STORAGE;


typedef struct UX_SLAVE_CLASS_STORAGE_PARAMETER_STRUCT
{
    ULONG                       ux_slave_class_storage_parameter_number_lun;
    UX_SLAVE_CLASS_STORAGE_LUN  ux_slave_class_storage_parameter_lun[UX_MAX_SLAVE_LUN];

} UX_SLAVE_CLASS_STORAGE_PARAMETER;



typedef struct UX_HOST_CLASS_CDC_ACM_STRUCT
{

    struct UX_HOST_CLASS_CDC_ACM_STRUCT
                    *ux_host_class_cdc_acm_next_instance;
    UX_HOST_CLASS   *ux_host_class_cdc_acm_class;
    UX_DEVICE       *ux_host_class_cdc_acm_device;
    UX_INTERFACE    *ux_host_class_cdc_acm_interface;
    UX_ENDPOINT     *ux_host_class_cdc_acm_bulk_out_endpoint;
    UX_ENDPOINT     *ux_host_class_cdc_acm_bulk_in_endpoint;
    UX_ENDPOINT     *ux_host_class_cdc_acm_interrupt_endpoint;
    UINT            ux_host_class_cdc_acm_state;
    TX_SEMAPHORE    ux_host_class_cdc_acm_semaphore;
} UX_HOST_CLASS_CDC_ACM;



typedef struct UX_HOST_CLASS_DPUMP_STRUCT
{

    struct UX_HOST_CLASS_DPUMP_STRUCT
                    *ux_host_class_dpump_next_instance;
    UX_HOST_CLASS   *ux_host_class_dpump_class;
    UX_DEVICE       *ux_host_class_dpump_device;
    UX_INTERFACE    *ux_host_class_dpump_interface;
    UX_ENDPOINT     *ux_host_class_dpump_bulk_out_endpoint;
    UX_ENDPOINT     *ux_host_class_dpump_bulk_in_endpoint;
    UX_ENDPOINT     *ux_host_class_dpump_interrupt_endpoint;
    UINT            ux_host_class_dpump_state;
    TX_SEMAPHORE    ux_host_class_dpump_semaphore;
} UX_HOST_CLASS_DPUMP;


typedef struct UX_SLAVE_CLASS_DPUMP_STRUCT
{
    UX_SLAVE_INTERFACE      *ux_slave_class_dpump_interface;

} UX_SLAVE_CLASS_DPUMP;
```

U S B X

APPENDIX D

Language Identifiers

The 16-bit language identifiers, and the Primary Language and Sublanguage Identifiers are defined in following sections.

16-bit Language Identifiers

The following are 16-bit language identifiers. They are composed of a 10-bit (9-0) Primary Language Identifier and a 6-bit (15-10) Sublanguage Identifier.

0x0436	Afrikaans
0x041c	Albanian
0x0401	Arabic (Saudi Arabia)
0x0801	Arabic (Iraq)
0x0c01	Arabic (Egypt)
0x1001	Arabic (Libya)
0x1401	Arabic (Algeria)
0x1801	Arabic (Morocco)
0x1c01	Arabic (Tunisia)
0x2001	Arabic (Oman)
0x2401	Arabic (Yemen)
0x2801	Arabic (Syria)
0x2c01	Arabic (Jordan)
0x3001	Arabic (Lebanon)
0x3401	Arabic (Kuwait)
0x3 801	Arabic (U.A.E.)
0x3c01	Arabic (Bahrain)
0x4001	Arabic (Qatar)
0x042b	Armenian.
0x044d	Assamese.
0x042c	Azeri (Latin)
0x082c	Azeri (Cyrillic)
0x042d	Basque
0x0423	Belarussian
0x0445	Bengali.
0x0402	Bulgarian
0x0455	Burmese
0x0403	Catalan
0x0404	Chinese (Taiwan)
0x0804	Chinese (PRC)
0x0c04	Chinese (Hong Kong SAR, PRC)
0x1004	Chinese (Singapore)
0x1404	Chinese (Macau SAR)
0x041a	Croatian
0x0405	Czech

0x0406	Danish
0x0413	Dutch (Netherlands)
0x0813	Dutch (Belgium)
0x0409	English (United States)
0x0809	English (United Kingdom)
0x0c09	English (Australian)
0x1009	English (Canadian)
0x1409	English (New Zealand)
0x1809	English (Ireland)
0x1c09	English (South Africa)
0x2009	English (Jamaica)
0x2409	English (Caribbean)
0x2809	English (Belize)
0x2c09	English (Trinidad)
0x3 009	English (Zimbabwe)
0x3409	English (Philippines)
0x0425	Estonian
0x0438	Faeroese
0x0429	Farsi
0x040b	Finnish
0x040c	French (Standard)
0x080c	French (Belgian)
0x0c0c	French (Canadian)
0x100c	French (Switzerland)
0x140c	French (Luxembourg)
0x180c	French (Monaco)
0x0437	Georgian.
0x0407	German (Standard)
0x0807	German (Switzerland)
0x0c07	German (Austria)
0x1007	German (Luxembourg)
0x1407	German (Liechtenstein)
0x0408	Greek
0x0447	Gujarati.
0x040d	Hebrew
0x0439	Hindi.
0x040e	Hungarian
0x040f	Icelandic
0x0421	Indonesian
0x0410	Italian (Standard)
0x0810	Italian (Switzerland)

0x0411	Japanese
0x044b	Kannada.
0x0860	Kashmiri (India)
0x043f	Kazakh
0x0457	Konkani.
0x0412	Korean
0x0812	Korean (Johab)
0x0426	Latvian
0x0427	Lithuanian
0x0827	Lithuanian (Classic)
0x042f	Macedonian
0x043e	Malay (Malaysian)
0x083e	Malay (Brunei Darussalam)
0x044c	Malayalam.
0x0458	Manipuri
0x044e	Marathi.
0x0861	Nepali (India).
0x0414	Norwegian (Bokmal)
0x0814	Norwegian (Nynorsk)
0x0448	Oriya.
0x0415	Polish
0x0416	Portuguese (Brazil)
0x0816	Portuguese (Standard)
0x0446	Punjabi.
0x0418	Romanian
0x0419	Russian
0x044f	Sanskrit.
0x0c1a	Serbian (Cyrillic)
0x081a	Serbian (Latin)
0x0459	Sindhi
0x041b	Slovak
0x0424	Slovenian
0x040a	Spanish (Traditional Sort)
0x080a	Spanish (Mexican)
0x0c0a	Spanish (Modern Sort)
0x100a	Spanish (Guatemala)
0x140a	Spanish (Costa Rica)
0x180a	Spanish (Panama)
0x1c0a	Spanish (Dominican Republic)
0x200a	Spanish (Venezuela)
0x240a	Spanish (Colombia)

0x280a	Spanish (Peru)
0x2c0a	Spanish (Argentina)
0x300a	Spanish (Ecuador)
0x340a	Spanish (Chile)
0x380a	Spanish (Uruguay)
0x3c0a	Spanish (Paraguay)
0x400a	Spanish (Bolivia)
0x440a	Spanish (El Salvador)
0x480a	Spanish (Honduras)
0x4c0a	Spanish (Nicaragua)
0x500a	Spanish (Puerto Rico)
0x0430	Sutu
0x0441	Swahili (Kenya)
0x041d	Swedish
0x081d	Swedish (Finland)
0x0449	Tamil.
0x0444	Tatar (Tatarstan)
0x044a	Telugu.
0x041e	Thai
0x041f	Turkish
0x0422	Ukrainian
0x0420	Urdu (Pakistan)
0x0820	Urdu (India)
0x0443	Uzbek (Latin)
0x0843	Uzbek (Cyrillic)
0x042a	Vietnamese
0x04ff	HID (Usage Data Descriptor)
0xf0ff	HID (Vendor Defined 1)
0xf4ff	HID (Vendor Defined 2)
0xf8ff	HID (Vendor Defined 3)
0xfcff	HID (Vendor Defined 4)

Primary Language Identifiers

Identifier	Predefined symbol	Language
0x39	LANG_HINDI	Hindi
0x3e	LANG_MALAY	Malay
0x3f	LANG_KAZAK	Kazak
0x41	LANG_SWAHILI	Swahili
0x43	LANG_UZBEK	Uzbek
0x44	LANG_TATAR	Tatar
0x45	LANG_BENGALI	Bengali
0x46	LANG_PUNJABI	Punjabi
0x47	LANG_GUJARATI	Gujarati
0x48	LANG_ORIYA	Oriya
0x49	LANG_TAMIL	Tamil
0x4a	LANG_TELUGU	Telugu
0x4b	LANG_KANNADA	Kannada
0x4c	LANG_MALAYALAM	Malayalam
0x4d	LANG_ASSAMESE	Assamese
0x4e	LANG_MARATHI	Marathi
0x4f	LANG_SANSKRIT	Sanskrit
0x57	LANG_KONKANI	Konkani
0x58	LANG_MANIPURI	Manipuri
0x59	LANG_SINDHI	Sindhi
0x60	LANG_KASHMIRI	Kashmiri
0x61	LANG_NEPALI	Nepali
0x62-0xfe		Reserved
0xff	LANG_HID	Reserved for USB HID Class use
0x1 00-0x3ff		Reserved

Sublanguage Identifiers

The following are sublanguage identifiers. They can be combined with Primary Language Identifiers to form Language Identifiers.

Identifier	Predefined symbol	Language
0x00-0x02		Reserved
0x01	SUBLANG_ARABIC_SAUDI_ARABIA	Arabic (Saudi Arabia)
0x02	SUBLANG_ARABIC_IRAQ	Arabic (Iraq)
0x03	SUBLANG_ARABIC_EGYPT	Arabic (Egypt)
0x04	SUBLANG_ARABIC_LIBYA	Arabic (Libya)
0x05	SUBLANG_ARABIC_ALGERIA	Arabic (Algeria)
0x06	SUBLANG_ARABIC_MOROCCO	Arabic (Morocco)
0x07	SUBLANG_ARABIC_TUNISIA	Arabic (Tunisia)
0x08	SUBLANG_ARABIC_OMAN	Arabic (Oman)
0x09	SUBLANG_ARABIC_YEMEN	Arabic (Yemen)
0x10	SUBLANG_ARABIC_SYRIA	Arabic (Syria)
0x11	SUBLANG_ARABIC_JORDAN	Arabic (Jordan)
0x12	SUBLANG_ARABIC_LEBANON	Arabic (Lebanon)
0x13	SUBLANG_ARABIC_KUWAIT	Arabic (Kuwait)
0x14	SUBLANG_ARABIC_UAE	Arabic (U.A.E.)
0x15	SUBLANG_ARABIC_BAHRAIN	Arabic (Bahrain)
0x16	SUBLANG_ARABIC_QATAR	Arabic (Qatar)
0x01	SUBLANG_AZERI_CYRILLIC	Azeri (Cyrillic)
0x02	SUBLANG_AZERI_LATIN	Azeri (Latin)
0x01	SUBLANG_CHINESE_TRADITIONAL	Chinese (Traditional)
0x02	SUBLANG_CHINESE_SIMPLIFIED	Chinese (Simplified)
0x03	SUBLANG_CHINESE_HONGKONG	Chinese (Hong Kong SAR, PRC)
0x04	SUBLANG_CHINESE_SINGAPORE	Chinese (Singapore)
0x05	SUBLANG_CHINESE_MACAU	Chinese (Macau SAR)
0x01	SUBLANG_DUTCH	Dutch
0x02	SUBLANG_DUTCH_BELGIAN	Dutch (Belgian)
0x01	SUBLANG_ENGLISH_US	English (US)
0x02	SUBLANG_ENGLISH_UK	English (UK)
0x03	SUBLANG_ENGLISH_AUS	English (Australian)
0x04	SUBLANG_ENGLISH_CAN	English (Canadian)
0x05	SUBLANG_ENGLISH_NZ	English (New Zealand)
0x06	SUBLANG_ENGLISH_EIRE	English (Ireland)

0x07	SUBLANG_ENGLISH_SOUTH_AFRICA	English (South Africa)
0x08	SUBLANG_ENGLISH_JAMAICA	English (Jamaica)
0x09	SUBLANG_ENGLISH_CARIBBEAN	English (Caribbean)
0x0a	SUBLANG_ENGLISH_BELIZE	English (Belize)
0x0b	SUBLANG_ENGLISH_TRINIDAD	English (Trinidad)
0x0c	SUBLANG_ENGLISH_PHILIPPINES	English (Zimbabwe)
0x0d	SUBLANG_ENGLISH_ZIMBABWE	English (Philippines)
0x01	SUBLANG_FRENCH	French
0x02	SUBLANG_FRENCH_BELGIAN	French (Belgian)
0x03	SUBLANG_FRENCH_CANADIAN	French (Canadian)
0x04	SUBLANG_FRENCH_SWISS	French (Swiss)
0x05	SUBLANG_FRENCH_LUXEMBOURG	French (Luxembourg)
0x06	SUBLANG_FRENCH_MONACO	French (Monaco)
0x01	SUBLANG_GERMAN	German
0x02	SUBLANG_GERMAN_SWISS	German (Swiss)
0x03	SUBLANG_GERMAN_AUSTRIAN	German (Austrian)
0x04	SUBLANG_GERMAN_LUXEMBOURG	German (Luxembourg)
0x05	SUBLANG_GERMAN_LIECHTENSTEIN	German (Liechtenstein)
0x01	SUBLANG_ITALIAN	Italian
0x02	SUBLANG_ITALIAN_SWISS	Italian (Swiss)
0x02	SUBLANG_KASHMIRI_INDIA	Kashmiri (India)
0x01	SUBLANG_KOREAN	Korean
0x01	SUBLANG_LITHUANIAN	Lithuanian
0x01	SUBLANG_MALAY_MALAYSIA	Malay (Malaysia)
0x02	SUBLANG_MALAY_BRUNEI_DARUSSALAM	Malay (Brunei Darassalam)
0x02	SUBLANG_NEPALI_INDIA	Nepali (India)
0x01	SUBLANG_NORWEGIAN_BOKMAL	Norwegian (Bokmal)
0x02	SUBLANG_NORWEGIAN_NYNORSK	Norwegian (Nynorsk)
0x01	SUBLANG_PORTUGUESE	Portuguese (Brazilian)
0x02	SUBLANG_PORTUGUESE_BRAZILIAN	Portuguese
0x02	SUBLANG_SERBIAN_LATIN	Serbian (Latin)
0x03	SUBLANG_SERBIAN_CYRILLIC	Serbian (Cyrillic)
0x01	SUBLANG_SPANISH	Spanish (Castilian)
0x02	SUBLANG_SPANISH_MEXICAN	Spanish (Mexican)
0x03	SUBLANG_SPANISH_MODERN	Spanish (Modern)
0x04	SUBLANG_SPANISH_GUATEMALA	Spanish (Guatemala)
0x05	SUBLANG_SPANISH_COSTA_RICA	Spanish (Costa Rica)
0x06	SUBLANG_SPANISH_PANAMA	Spanish (Panama)
0x07	SUBLANG_SPANISH_DOMINICAN_REPUBLIC	Spanish (Dominican Republic)
0x08	SUBLANG_SPANISH_VENEZUELA	Spanish (Venezuela)

0x09	SUBLANG_SPANISH_COLOMBIA	Spanish (Colombia)
0x0a	SUBLANG_SPANISH_PERU	Spanish (Peru)
0x0b	SUBLANG_SPANISH_ARGENTINA	Spanish (Argentina)
0x0c	SUBLANG_SPANISH_ECUADOR	Spanish (Ecuador)
0x0d	SUBLANG_SPANISH_CHILE	Spanish (Chile)
0x0e	SUBLANG_SPANISH_URUGUAY	Spanish (Uruguay)
0x0f	SUBLANG_SPANISH_PARAGUAY	Spanish (Paraguay)
0x10	SUBLANG_SPANISH_BOLIVIA	Spanish (Bolivia)
0x11	SUBLANG_SPANISH_EL_SALVADOR	Spanish (El Salvador)
0x12	SUBLANG_SPANISH_HONDURAS	Spanish (Honduras)
0x13	SUBLANG_SPANISH_NICARAGUA	Spanish (Nicaragua)
0x14	SUBLANG_SPANISH_PUERTO_RICO	Spanish (Puerto Rico)
0x01	SUBLANG_SWEDISH	Swedish
0x02	SUBLANG_SWEDISH_FINLAND	Swedish (Finland)
0x01	SUBLANG_URDU_PAKISTAN	Urdu (Pakistan)
0x02	SUBLANG_URDU_INDIA	Urdu (India)
0x01	SUBLANG_UZBEK_LATIN	Uzbek (Latin)
0x02	SUBLANG_UZBEK_CYRILLIC	Uzbek (Cyrillic)
0x01	SUBLANG_HID_USAGE_DATA_DESCRIPTOR	HID (Usage Data Descriptor)
0x3c	SUBLANG_HID_VENDOR_DEFINED_1	HID (Vendor Defined 1)
0x3d	SUBLANG_HID_VENDOR_DEFINED_2	HID (Vendor Defined 2)
0x3e	SUBLANG_HID_VENDOR_DEFINED_3	HID (Vendor Defined 3)
0x3f	SUBLANG_HID_VENDOR_DEFINED_4	HID (Vendor Defined 4)

U S B X

H

I

J

L

M

O

P

R

S

T

U

V

W

U S B X