Table Of Contents

Document information

History

V0.1 ¸- Initial draft.

V0.2 - Fixed USB API of the System Software to an Exec-style device.
Send/Receive now done using IoRequests, allowing both Syncronous and Asyncronous operation without USB System intervention.
Added specification of USBClaimFunction(fkt,NULL), allowing expunging lockout on Function references.
USBSetAddress() and USBSyncFrame() function calls excluded from interface, as they are only for Host Controller Driver use. The individual HCDs must implement these internally, allowing HCDs to do these things in an implementation specific manner.

V0.3 - USBClaimFunction(fkt,NULL) replaced by USBLockFunction(), and USBDeclaimFunction(fkt,NULL) replaced by USBUnlockFunction(). This is done to clarify function call usage.
Added <fdfkt> argument to USBClaimFunction() to tie a Function Drivers private USB Function reference. This in turn allows the USS to give this reference with notification messages, informing the FD which USB Function it is sending notification for.

Disclaimer

USB System API

The following sections describe a proposal for the API between the USB System Software and the USB Function drivers. The API abstracts the USB bus protocols by using a more simple function-call interface.
It is not the intension of this document to specify the inner workings of the USB System Software, although some hints are given thru function arguments and the occasional implementation note.
In the following references are made to descriptors. Two types of descriptors exist - USB System Software descriptors which are structures used internally by the System Software for working with USB and used by the client software as arguments when calling System Software functions, and USB descriptors which are the descriptors held in, and returned by, USB Functions thru the Default Control Pipe.
The text should be cleared up as two which are referenced by the word descriptor. Possibly another word should be used for the USB System Software structures to clarify things.

Data flow

After having setup a USB device by selecting its configuration etc. data is sent back and forth to the USB Function using two API functions, namely USBSend() and USBReceive(). These two function calls are in fact exec device messages sent to the USB System Software (the USB System Function driver API is an Exec style device as seen from the Function driver).

USBSend

LONG USBSend(endpoint, data, size);

Ths function sends a block of data to an EndPoint of a USB Function. The data block given to this function needs not be of the EndPoint's specified max. data packet size, as this function will internally split transfers into multiple packets if necessary. The data packet size rules of the EndPoint must, however, still be obeyed. The data block must therefore, if the EndPoints requires it, be a multiple of the EndPoint's max. data packet size.
This function returns an error code, indicating the result of the data transmission.

endpoint	-	USB System private reference to the endpoint to send data to. This is the reference returned by a call to USBGetEndPoint().
data	-	Pointer to the data to send.
size	-	Bytelength of the data to send.

USBReceive

LONG USBReceive(endpoint, data, size)

This function requests data retrieval from an EndPoint of a USB Function. Up to

<size> bytes of data will be read into the specified data buffer from the EndPoint. This function will transparently break up the request into smaller USB packets to meet the max. packet size of the EndPoint. Any constraints on packet size for the EndPoint must, however, still be obeyed. The data buffer must therefore, if the EndPoint requires it, be a multiple of the EndPoint's max. data packet size.
This function returns an error code indicating the result of the data reception.

endpoint	-	USB System private reference to the endpoint to receive data from. This is the reference returned by a call to USBGetEndPoint().
data	-	Pointer to a buffer for placing received data into.
size	-	Bytelength of the data buffer.

USB Function control

To be able to locate and work with USB Functions a set of API functions are defined.

USBFindFunction

usfkt = USBFindFunction(class, subclass, current)

This function is used to locate a USB Function attached to the system. The Function is found based on its USB class and subclass.
To be able to traverse through some, or all, USB Functions of a specified class and/or subclass this function takes a <current> argument. This is a pointer to the current USB Function descriptor being looked into, and searching will commence at the next USB Function descriptor. For first-time searches a NULL current-Function will initiate search from the top of the internal Function list.
This function will either return a reference to the next USB Function descriptor with the specified class and/or subclass, or NULL if no matching descriptor was found. The returned reference is USB System private, and must only be used as argument to other USB System calls, not peeked into.
The returned USB Function reference has been USBLockFunction()'ed, ensuring that the Function cannot be expunged from the system. As a consequence, all USB Function references returned by this function MUST be USBUnlockFunction()'ed after use (unless you do a USBRemFunction() which itself unlocks the Function).

class	-	USB Class code of Function to search for. Specify -1 for any class
subclass	-	USB Subclass code (Class specific) of Function to search for. Specify -1 for any subclass.
current	-	Pointer to USB Function descriptor after which to initiate searching. NULL initiates search at the beginning of the internal descriptor list.

USBLockFunction

void USBLockFunction(usfkt)

With this function you lock a USB Function into place in the USS. While locked a USB Function cannot be expunged by the USS, ensuring that the USB Function reference you hold is not pulled away from under your feet.
Normally locking of USB Functions is handled by the USS before returning a Function reference.
USBLockFunction maintains an internal lock count, allowing nesting calls. For every call to USBLockFunction() a matching call must be made to USBUnlockFunction().

Note that while this function will lock a USB Function from being expunged it does not grant exclusive ownership of a USB Function. Use USBClaimFunction() for that.

usfkt - USB System private reference to the Function to lock. This is a reference as returned by USBFindFunction().

USBUnlockFunction

void USBUnlockFunction(usfkt)

This function performs a single unlock one USB Function, undoing the lock placed on the Function by a call to USBLockFunction().
All USBLockFunction() calls must be matched by a call to USBUnlockFunction() to allow expunging of the USB Function from the USS if the physical USB Function is removed from the USB bus.

usfkt - USB System private reference to the Function to unlock. This is a reference as returned by USBFindFunction().

USBClaimFunction

ULONG USBClaimFunction(usfkt, fdfkt, port)

With this function a USB Function driver claims ownership of a USB Function. Only one function driver may own a USB Function at any time.
Claiming a USB Function links a standard Exec MsgPort with the USB Function. This MsgPort will be used for information passing from the USB System software to the device driver. This includes information such as detachment notification etc.
This function will return NULL if the USB Function could not be claimed (function was already claimed). Otherwise a non-NULL claim key is returned. This key must be used when de-claiming the Function at a later time.
All calls to USBClaimFunction() must be matched with a call to USBDeclaimFunction() at a later time.
Note that this function is *not* nesting. A single call to USBDeclaimFunction() will undo any number of calls to USBClaimFunction().

usfkt	-	USB System private reference to the Function to claim. This is a reference as returned by USBFindFunction().
fdfkt	-	Claimer's private reference for the claimed Function. This will be returned with every notification message sent to the claimer, allowing it to identify the Function which the notification is for (allowing a FD to handle more than one USB Function).
port	-	Pointer to a MsgPort thru which to receive msgs regarding the Function.

USBDeclaimFunction

void USBDeclaimFunction(usfkt, key)

This function is used for de-claiming a USB Function after use. When this function returns the given Function is nolonger bound to a MsgPort, and no driver owns the Function any longer.
Usually this function is used in conjunction with the refered USB Function being physically unplugged from the USB topology. When that happens the Function driver is informed thru a notification message. Once the driver has cleaned up up after itself (it nolonger being required when the USB Function has been removed) the driver declaims the USB Function. This, in turn, allows the USB System software to free any ressources consumed for the removed USB Function, including the USB Function descriptor.
To ensure that only the rightfull owner of the USB Function (the one who claimed the Function in the first place) can declaim the USB Function , the claim key returned by USBClaimFunction() must be given as argument. If the claim key is incorrect this function does nothing.

usfkt	-	USB System private reference to the Function to claim. This is a reference as returned by USBFindFunction().
key	-	Claim key returned by USBClaimFunction() at claim time.

USBAddFunction

usfkt = USBAddFunction(ushub, taglist)

This function is used for adding a new USB Function to the USB System software. When a hub in the USB topology detects insertion of a USB Function, the hub driver can append the inserted USB Function to the USB System software using this function. All information about the new USB Function is supplied using a tag list. This abstracts the USB Function descriptor from the hub driver, allowing easier extensions to the USB Function descriptor in the future.
This function returns a USB System Software private reference to the USB Function appended by this function call. The hub driver should keep a record of this reference

to allow removal of the USB Function if it is unplugged from the hub at a later time. To allow safe use of the returned <usfunction> reference the Function has been USBLockFunction()'ed to keep it from being expunged.

ushub - USB System private reference to the USB Hub Function the new Function has been inserted into. This is a reference as returned by USBFindFunction().
taglist - Taglist holding information on the new USB Function.

USBRemFunction

USBRemFunction(usfkt)

This function removes a USB Function from the USB System software. That is, the function descriptor is marked as unavailable, keeping it from being seen on Function searches, but allowing the current Function owner to still access the descriptor until USBDeclaimFunction() is called by the owner (after which the Function is expunged from the System Software).
This function will do a USBUnlockFunction() on the referred Function, undoing the USBLockFunction() call made by a USBAddFunction() or USBFindFunction() call.
The intended use of this function is for USB hub drivers, when a USB Function is unplugged from one of the hub ports.
On an internal note, the removed USB Function descriptor is not expunged from memory until the current owner of the USB Function has de-claimed the Function, and all locks are unlocked.

usfkt - USB System private reference to the USB Function which has been removed from the USB topology (unplugged). This is a reference as returned by USBFindFunction().

USBGetEndPoint

usep = USBGetEndPoint(usfkt, endpointID)

This function returns the USB System Software reference for an EndPoint in a USB Function. The EndPoint is located by its EndPoint number, as dictated by the current configuration of the USB Function (as set with USBSetConfiguration()).
USBGetEndPoint() will return the Default Control Pipe EndPoint if endpointID is zero.
If no EndPoint exist with the given EndPoint number this function returns NULL.

Only the owner of the USB Function in question is allowed to call USBGetEndPoint(). Use USBClaimFunction() to gain ownership.

usfkt - USB System private reference for the Function to get an EndPoint from.

endpointID - The number of the EndPoint to get a reference to. This value corresponds to the EndPoint number part of the bEndpointAddress field of the USB Function Standard Endpoint Descriptor (see USB spec V1.1 p. 203). Use USBGetDescriptor() to access standard descriptors of a USB Function.

USB Function abstraction

These are function call representations of standard USB Default Control Pipe commands. Currently not all are fully specified, but ideally they should be, functioning as a USB Function wrapper for the standard Default Control Pipe command set.

USBGetDescriptor

LONG USBGetDescriptor(usfunction, type, index, buffer, size, taglist)

This function queries a USB Function for a descriptor. The type of descriptor, and the index into the descriptor array for that type of descriptor, must be specified for the USB Function to be able to return the correct descriptor. The descriptors obtainable are those specified in the USB Specification version used by the referenced USB Function.
This function is a representation of the USB Default Control Pipe GetDescriptor() command.
Some descriptor types may need more information than an index number. Such extra arguments must be specified using tags in the TagList.
This function returns the number of descriptor databytes read into the buffer, or zero if the descriptor could not be obtained. If the returned byte count equals the size of the buffer it is possible that not all descriptor databytes were read due to lack of buffer space. The caller must be prepared to handle such a situation by verifying if all data was received (by evaluating the descriptor's size fields). If not all descriptor data was received a new (larger) buffer must be obtained and the USBGetDescriptor() call retried.

Normally this function will only be used in conjunction with USB Function configuration. As a consequence the USB Function in question needs not be claimed to use this function.

NOTE: Shold this function instead internally allocate a buffer big enough, and then be matched by a USBFreeDescriptor() call, instead of the caller having to supply a buffer?? - No, this call is open ended and allows support of descriptors unknown to the USB System software. It puts more effort at the application, but is more flexible!

usfunction - USB System private reference to the Function to obtain a descriptor from. This is a reference as returned by USBFindFunction().
type - The type of descriptor to obtain (see USB spec. V1.1 p.187)
index - The index number of the descriptor to obtain. Zero based.

USBGetConfiguration

LONG USBGetConfiguration(usfunction, &buffer)

USBGetInterface

LONG USBGetInterface(usfunction, interfaceID, &buffer)

USBGetStatus

LONG USBGetStatus(usfunction, type, &buffer)

USBSetAddress

LONG USBSetAddress(usfunction, address)

NOTE: Should this function exist at all?? Isn't the USB address of a USB Function totally under Host Controller Driver control? I should think so. No-one but the HCD needs to know the USB bus address of a Function. The HCD can then implement address assignment as it best suits itself

USBSetConfiguration

LONG USBSetConfiguration(usfunction, configID)

Using this function call you select the Configuration used by the specified USB Function. When changing Configuration you also change the way EndPoints are used in the USB Function. The USB System Software therefore clears any previously used EndPoints upon selecting a Configuration. This behavior also mean that after calling this function any EndPoint descriptor references obtained from the Function must nolonger be used. Any requests pending at old EndPoints will also be aborted.
This function returns an error code, indicating the result of trying to change configuration.

usfunction	-	USB System private reference to the Function to change Configuration of. This is a reference as returned by USBFindFunction().
configID	-	The number of the Configuration to use. Specify zero to clear Configuration and place the USB Function in its Address state.

USBSetDescriptor

LONG USBSetDescriptor(usfunction, type, index, buffer, size, taglist)

USBSetFeature

LONG USBSetFeature(usfunction, type, feature)

USBClearFeature

LONG USBClearFeature(usfunction, type, feature)

USBSetInterface

LONG USBSetInterface(usfunction, interfaceID, setting)

USBSyncFrame

LONG USBSyncFrame(usendpoint, &buffer)

NOTE: Should this function exist at all?? Applications should not be involved in tampering with frame syncronization. This should be left under HCD control. It also allows the HCD to do syncronization as it best suits itself.