AN1144

USB Human Interface Device Class on an Embedded Host

Author: *Amardeep Gupta*
Microchip Technology Inc.

INTRODUCTION

With the introduction of Microchip's microcontrollers with the USB OTG peripheral, microcontroller applications can easily support USB embedded host functionality. Traditionally, the PC is used as a host in an USB network. Now, with Microchip's microcontroller with host capability, the host can be implemented in an embedded system. Some of the most common uses of this capability are to interface to Human Interface Devices (HIDs).

USB HUMAN INTERFACE DEVICE (HID) CLASS

Overview

The HID class primarily consists of devices that are used to control any particular application.

Typical examples of HID class devices include:

- Keyboard and pointing devices
- Control switches, sliders and so on
- Joystick, steering and other gaming control inputs
- Point-of-sale bar code scanners and magnetic card readers having an HID Keyboard Emulation mode

The HID class can be used for devices without human interface, too; such applications just need to be able to function within the limits of the HID class specifications.

Key Features of HID Class

- Data is exchanged between the host and the device in the form of reports. The format of the report is defined by the report descriptor defined by the device based on device need.
- An HID interface uses Interrupt Transfer mode to move the data.
- An HID interface must have at least one interrupt IN endpoint for sending the input report. The HID interface also has an optional interrupt OUT endpoint. If the interrupt OUT endpoint is not defined, the output report can be sent over the control OUT endpoint. However, the application must ensure that the transfers over the control endpoint are not frequent.
- As an HID class uses Interrupt Transfer mode, a maximum of 64 bytes can be transferred in a single frame (i.e., 64 Kbyte/s per endpoint when operating in Full-Speed mode).

The class, subclass and protocol designators for an HID device are not contained in the `bDeviceClass`, `bDeviceSubClass` and `bDeviceProtocol` fields of the device descriptor. Instead, these fields are all set to 0x00 and the designators are specified in the `bInterfaceClass`, `bInterfaceSubClass` and `bInterfaceProtocol` fields of the interface descriptor. The most common configurations for HID class devices are:

- `bInterfaceClass` –
  0x03 (HID Class)
- `bInterfaceSubClass` –
  0x00 (No Subclass)
  0x01 (Boot Interface Subclass)
  0x02-0xFF (Reserved)
- `bInterfaceProtocol` –
  0x00 (None)
  0x01 (Keyboard)
  0x02 (Mouse)
  0x03-0xFF (Reserved)

A host communicates with the HID class device using either the control (default) pipe or an interrupt pipe.

The control pipe is used for:

- Sending and receiving the control transfer data.
- Transmitting and receiving reports if the interrupt endpoint is not used by the device.

The interrupt pipe is used for:

- Transmitting and receiving reports to and from the device.
- Transmitting fixed latency data to or from the device.

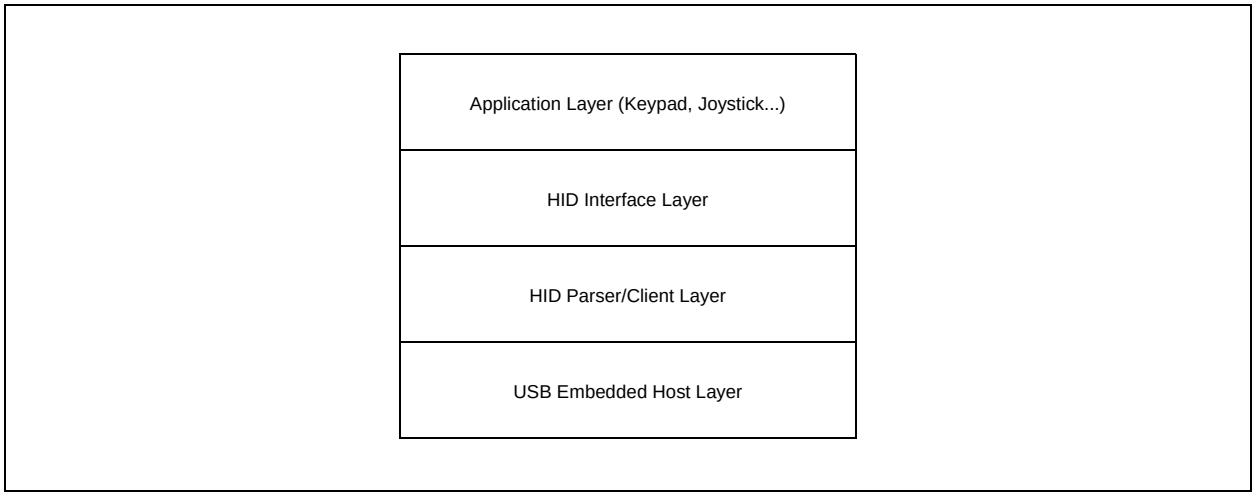
The interrupt OUT pipe is optional. If a device declares an interrupt OUT endpoint, then the output reports are transmitted by the host to the device through the interrupt OUT endpoint. If no interrupt OUT endpoint is declared, then the output reports are transmitted to a device through the control endpoint.

THE HID CLIENT DRIVER

Architecture of HID Client Driver

Applications do not interface directly with the USB HID client driver. Instead, they use an application interface layer which will interface with the client driver, which in turn, will use the host layer. Figure 1 displays the USB HID host application interface functions.

FIGURE 1: USB HID HOST ARCHITECTURE



The HID interface layer comprises the HID parser interfaces and interface functions to send and receive reports to and from the device. The report descriptor is parsed and the data is stored in predefined structures. The parser will populate these data structures with information extracted from the report descriptor. The application can use the functions defined in the interface layer to understand the report descriptor.

Note:	For detailed information about the USB host HID driver API and HID parser, refer to the API documentation provided in the Help directory.

USING THE HID CLIENT DRIVER

This section provides a brief overview of the installation and configuration procedures. For detailed information on installation and configuration, refer to *AN1140, "USB Embedded Host Stack"* and *AN1141, "USB Embedded Host Stack Programmer's Guide"*.

Installing the HID Client Driver

The HID client driver is installed as part of the complete USB embedded host support package, available on the Microchip web site (http://www.microchip.com/usb).

Configuring the USB HID Class

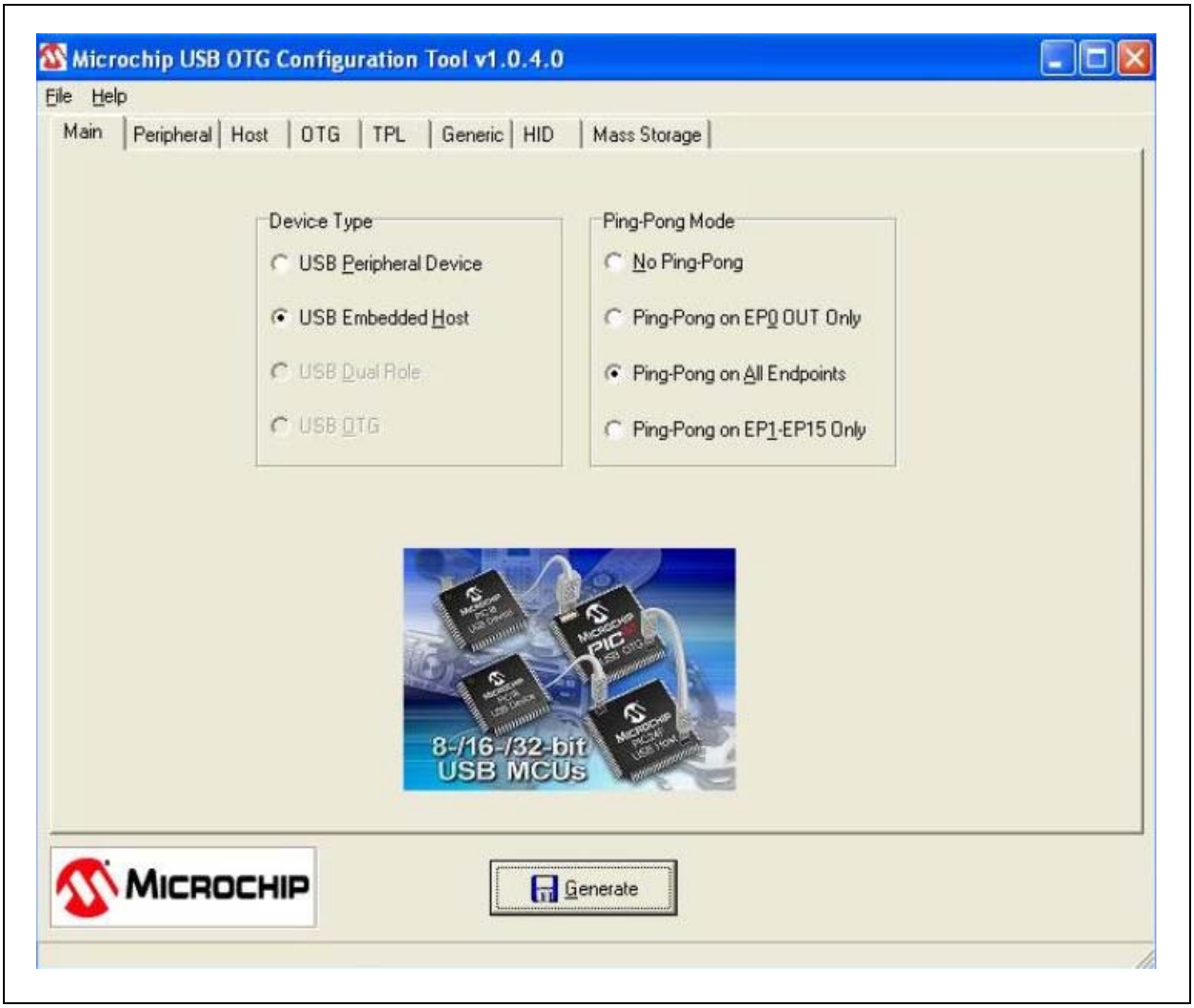
Use the configuration tool, `USBConfig.exe`, to configure the HID client driver for an application. This tool is installed in the `.\Microchip\USB` directory.

The following sections provide a brief description about the configuration of `USBConfig.exe`.

Main Tab

To use the HID client driver for a USB embedded host, the **USB Embedded Host** radio button in the Main tab will be selected by default, as displayed in Figure 2.

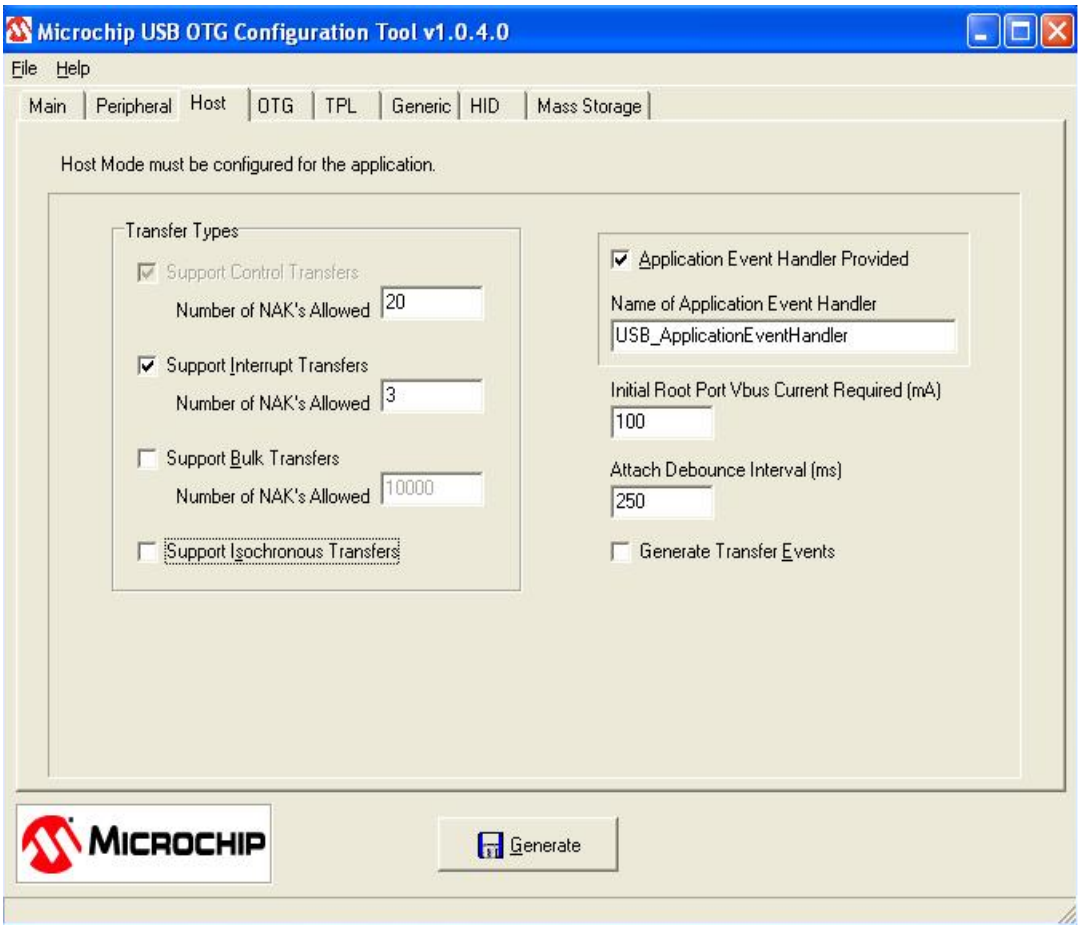
FIGURE 2: USB CONFIGURATION – MAIN TAB



Host Tab

1. Click the Host tab to configure basic host operation, as displayed in Figure 3.

FIGURE 3: USB CONFIGURATION – HOST TAB



The HID client driver requires support for control and interrupt endpoints.

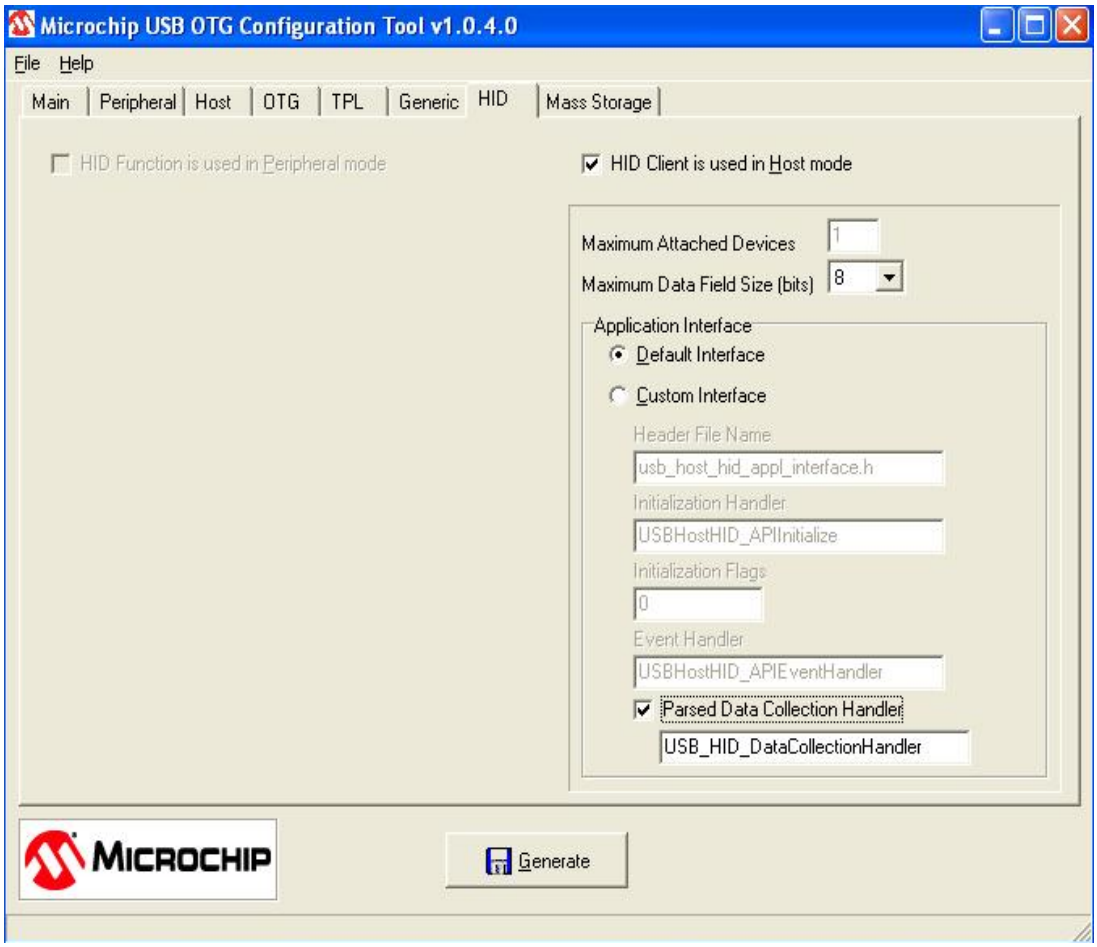
2. Under **Transfer Types**, select the **Support Interrupt Transfers** check box and enter the required NAKs in the text box. Even though Full-Speed USB mode supports a 1 ms communication frame, the fastest transfer in most HID applications is at a rate of 10 ms. If the report from the device is unavailable, the device NAK is the response received by the host. As the Idle rate implemented on the device can vary from 0.004 sec to 1.020 sec, users should configure the **Number of NAKs Allowed** field in sync with the implementation on the device end. Unselect the **Support Bulk Transfers** and **Support Isochronous Transfers** if the application does not contain classes that require bulk or isochronous endpoints.
3. Some devices also require longer than the USB specification of 100 ms to initialize after power-up; it is recommended to increase the attach debounce interval.
4. Enter the name of the function in the main source file that serves as the application level event handler.
5. Select the **Generate Transfer Events** check box to utilize transfer events (`EVENT_TRANSFER`) from the USB host layer. Refer to the **"Event Generation"** section for more information about transfer events.

HID Tab

The USB HID client driver can either poll the USB host driver for transfer status or respond to the USB host driver transfer events.

1. Select the HID tab.
2. Select the **HID Client is used in Host mode** check box to enable support for an HID embedded host, as displayed in Figure 4.

FIGURE 4: USB CONFIGURATION – HID TAB



The HID report from the device can contain multiple data fields of varied length in bits.

3. Enter the required value in the **Maximum Data Field Size (bits)** text box. This value informs the HID client driver of the maximum size of the data field received from the device within a report.
4. Select the **Default Interface** radio button to use the default application interface.

To customize the application interface, select the **Custom Interface** radio button and provide the initialization and event handler functions.

The user application has to collect the information stored by the HID parser. The configuration tool provides an edit box to enter the function to collect this information. If the parser information is not required by the user application, leave the check box unselected.

Note:	Refer to *AN1212 "Using USB Keyboard with an Embedded Host"* for implementation details.

Defining the Callback Handlers

The client driver requires two callback handlers in the interface layer.

Initialization Event Handler – This is called after the peripheral has been enumerated and initialized by the host layer. The initialization handler should be of the type defined by the `typedef`:

```
typedef BOOL (*USB_CLIENT_INIT) (BYTE
address, DWORD flags);
```

This function performs initialization specific to the device. If initialization occurs with no error, this routine should return TRUE; otherwise, this routine should return FALSE. No transfers to the peripheral will be allowed.

Event Handler – This is required to handle events that occur during normal operation. This event handler should be of the type defined by the `typedef`:

```
typedef BOOL (*USB_CLIENT_EVENT_HANDLER)
(BYTE address, USB_EVENT event, void*data,
DWORD size);
```

For example, the event, `EVENT_DETACH`, occurs when a device has detached from the bus. In this case, the interface layer will need to update its status by doing operations, such as removal of the device from its list of attached devices. See API documentation provided in the Help directory for a complete list of events.

The client driver requires a list of the interface's required peripheral initialization and event handlers. This list is defined by the configuration tool, `USBConfig.exe`, provided with the stack.

EVENT GENERATION

The client driver can be configured to utilize transfer events (`EVENT_TRANSFER`) from the USB host layer. In addition, the client driver can be configured to generate transfer events (`EVENT_HID_TRANSFER`) for the interface layer. These two events can be configured independently of each other, giving four possible combinations.

Table 1 lists the available combinations.

TABLE 1: EVENT CONFIGURATIONS

USB Host Layer	USB Host HID Driver
Poll for transfer status	Poll for HID transfer status
Poll for transfer status	Generate HID transfer events
Generate transfer events	Poll for HID transfer status
Generate transfer events	Generate HID transfer events

If the USB embedded host transfer events are used:

- The application would require more program and data memory, but the application processing will be more efficient. The USB embedded host transfer event configuration is transparent to the interface layer.
- More program memory is required and the interface layer that handles these events must be structured properly. In general, the code architecture required to utilize the transfer events is more sophisticated. It may be more difficult for C programming learners to design, develop, debug and maintain.

The choice of whether or not to utilize the USB embedded host HID transfer events can also depend on the implementation of other layers in the application.

Note 1: Although the USB embedded host uses USB interrupts, transfer event generation from the host driver layer to the client driver is triggered by a polling mechanism. This is to ensure that the USB Interrupt Service Routine (ISR) completes in a timely fashion. For more information on the host driver, refer to *AN1140, "USB Embedded Host Stack"* and *AN1141, "USB Embedded Host Stack Programmer's Guide"*.

2: Regardless of whether or not USB embedded host HID transfer events are used, the interface layer is required to contain an event handler that processes other system events.

CLIENT DRIVER INITIALIZATION

The USB configuration tool provides a macro, `USBInitialize( )`, to call all of the initialization routines required by the USB embedded host layer and the supported client drivers.

Normal Client Driver Operation

Normal background operation is performed by a single function:

```
void USBHostHIDTasks(void);
```

This routine must be called on a regular basis to allow device operation. The polling rate is not critical, since most of the actual transfer of information is handled through the USB interrupt. Since an application may support multiple classes, this function does not call the USBHostTasks() function, which also must be called on a regular basis.

The USB configuration tool will provide the `USBTasks()` macro to call all of the background task routines required by the USB host driver and the supported client drivers.

Once the device is detected, the host layer enumerates the device and calls back the HID client layer to initialize the interfaces. The HID client layer then requests the report descriptor. Each item in the report descriptor is parsed, and only if it is found in proper format, the device is listed; otherwise, an error is flagged and the device is not attached.

Note:	For parser related information, refer to "*Device Class Definition for Human Interface Devices (HID)*", http://www.usb.org.

The HID parser parses the report descriptor from the device and fills in predefined data structures with the data extracted from the report descriptor. The HID parser extracts this information using a two-pass compilation.

The HID host driver supports multiple interfaces on a single device. There must be at least one report descriptor in a device. Once the report descriptor is validated by the driver, the data structures (see the **"HID Parser Details"** section) are populated and the `EVENT_HID_RPT_DESC_PARSED` event is triggered.

Note:	The `EVENT_HID_RPT_DESC_PARSED` event is triggered even if the user has not enabled transfer events. Refer to *AN1212, "Using USB Keyboard with an Embedded Host"* for implementation details.

The user application must provide a function to collect the parser information. The configuration tool has a provision to enter the function name that would collect the parser details. The application must create an structure of type, `HID_DATA_DETAILS`. The information required to fill this structure is present in the parsed data. The application can use the functions provided in the `usb_client_hid_appl_interface.c` file to fill the details. If the application does not define the function on the `EVENT_HID_RPT_DESC_PARSED` event, this event will return TRUE.

The HID host driver will assume that the application is aware of the report details and does not require the parsed data. The parsed data is lost after the event and is overwritten by the new report descriptor (in case of multiple interfaces). This is done to reduce dynamic RAM usage.

The `usb_host_hid_appl_interface.c` file defines the following functions required by the application:

- `USBHostHID_ApiGetReport()` – This function is used to receive the report from the device (see Example 1).
- `USBHostHID_ApiSendReport()` – This function is used to transmit the report (see Example 2).

If an endpoint is not defined to send the output report to the device, the Endpoint 0 (control transfer) will be used. The rest of the application communication would use the interrupt transfers. All of the report transfers must be initiated by the application. Once the device is enumerated, all the transfers must be scheduled by the user's application using the interface functions mentioned above.

Note:	Memory allocation for descriptor related information is dynamic. Report descriptor data structures consume almost 300 bytes per interface. The users should reserve 1 Kbyte of heap while using the HID host stack for their application.

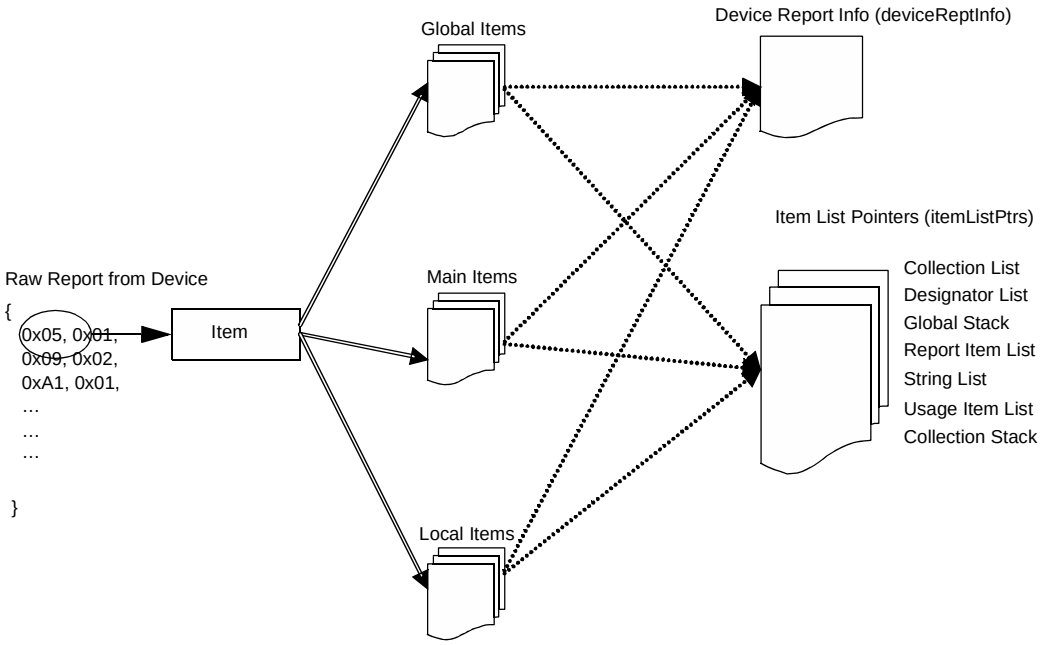
HID PARSER DETAILS

The HID parser, during the first pass through the report descriptor, counts various items to determine the sizes of the various tables within the pre-parsed report and the total memory that is required for the structure.

Once the memory has been allocated for the data structures and its tables, a second pass through the report descriptor is made to populate the tables. As shown in Figure 5, the HID parser picks each item from the raw report, categorizes it as either a global, main or local item, and creates a new entry in the buffers, “deviceReptInfo” and “itemListPtrs”. As each item is parsed, it might result in a new table entry, an updated table entry or an updated temporary variable.

Note:	For parser related information, refer to *“Device Class Definition for Human Interface Devices (HID)”,* http://www.usb.org.

FIGURE 5: HID PARSER



Note: Refer to *“Device Class Definition for Human Interface Devices (HID)”* for more details on global, main and local item details.

Parsing Main Items

Main Items processed are the Report Items Input, Output and Feature, and the Collection Items Collection and EndCollection. With the exception of EndCollection, parsing a main item always results in a new table entry.

Parsing Local Items

Local items processed that result in new table entries are Usage, StringIndex and DesignatorIndex. UsageMinimum, UsageMaximum, StringMinimum, StringMaximum, DesignatorMinimum and DesignatorMaximum work in pairs to generate new table entries; set delimiters are not supported.

Parsing Global Items

Global Items update temporary variables, but in most cases, do not directly affect table entries. An exception is the ReportID that may result in a new table entry.

The `USB_HID_ITEM_LIST` structure contains the pointers to arrays of item structures. Each collection entry is listed and described in the following individual tables.

Note:	All the data types are defined in the `GenericTypedefs.h` file.

Collection Entry Structure (`HID_COLLECTION`)

TABLE 2: COLLECTION ENTRY STRUCTURE (`HID_COLLECTION`)

Data Type	Data Field	Description
DWORD	data	This is the data from the collection item in the report descriptor.
WORD	usagePage	This is the usage page in effect when the collection item was parsed.
BYTE	firstUsageItem	This is the index of the first usage item associated with this collection (all associated usage items precede the collection item in the descriptor).
BYTE	usageItems	This is the number of associated usage items.
BYTE	firstReportItem	This is the index of the first report item (input/output/feature) contained within this collection.
BYTE	reportItems	This is the number of report items in this collection.
BYTE	parent	This is the index of the parent collection.
BYTE	children	This is the number of child collections for this collection.
BYTE	firstChild	This is the index of the “first” child collection in a link list of collections. The next child is indicated by the NextSibling in that child’s entry. Note that the children appear in the linked list in backwards order relative to their order in the descriptor.
BYTE	NextSibling	This is the index of the next sibling collection (or 0).

Report Entry Structure (`HID_REPORT`)

TABLE 3: REPORT ENTRY STRUCTURE (`HID_REPORT`)

Data Type	Data Field	Description
WORD	reportID	This is the ID of this report (or 0 for the default).
WORD	inputBits	This is the number of input bits associated with this report ID.
WORD	outputBits	This is the number of output bits associated with this report ID.
WORD	featureBits	This is the number of feature bits associated with this report ID.

Report Item Entry Structure (`HID_REPORTITEM`)

TABLE 4: REPORT ITEM ENTRY STRUCTURE (`HID_REPORTITEM`)

Data Type	Data Field	Description
HIDReportTypeEnum	reportType	This is the report type (input/output/feature) for this item.
HID_GLOBALS	globals	This structure contains the global items pertaining to current report.
BYTE	startBit	This is the Start bit of the report item in the report.
BYTE	parent	This is the index of the parent collection.
DWORD	dataModes	This is the data byte of the report item, indicating data modes such as Variable, Array and Relative.
BYTE	firstUsageItem	This is the index of the first usage item associated with this report item (all associated usage items precede the report item in the descriptor).
BYTE	usageItem	This is the number of associated usage items.
BYTE	firstStringItem	This is the index of the first string item associated with this report item (all associated string items precede the report item in the descriptor).
BYTE	stringItems	This is the number of associated string items.
BYTE	firstDesignatorItem	This is the index of the first string item associated with this report item (all associated designator items precede the report item in the descriptor).
BYTE	designatorItems	Number of associated designator items.

Usage Item Entry Structure (`HID_USAGEITEM`)

TABLE 5: USAGE ITEM ENTRY STRUCTURE (`HID_USAGEITEM`)

Data Type	Data Field	Description
BOOL	isRange	This indicates whether the item is a usage or a range of usages.
WORD	usagePage	This is the usage page for the item.
WORD	usage	This is the usage for a single usage.
WORD	usageMinimum	This is the minimum for a range of usage.
WORD	usageMaximum	This is the maximum for a range of usage.

String Item Entry Structure (`HID_STRINGITEM`)

TABLE 6: STRING ITEM ENTRY STRUCTURE (`HID_STRINGITEM`)

Data Type	Data Field	Description
BOOL	isRange	This indicates whether the item is a string index or a range of string indices.
WORD	index	This indicates a single index.
WORD	minimum	This is the minimum for a range of usage.
WORD	maximum	This is the maximum for a range of usage.

Designator Item Entry Structure (`HID_DESIGITEM`)

TABLE 7: DESIGNATOR ITEM ENTRY STRUCTURE (`HID_DESIGITEM`)

Data Type	Data Field	Description
BOOL	isRange	This indicates whether the item is a designator index or a range of string designators.
WORD	index	This is for a single index.
WORD	minimum	This is the minimum for a range of usage.
WORD	maximum	This is the maximum for a range of usage.

The application can import these data structures by including the `usb_host_hid_appl_interface.h` file.

The parser populates these data structures by extracting the information from the report descriptor. The user application can use the `USBHostHID_ApiFindBit()` and `USBHostHID_ApiFindValue()` functions to extract information or can traverse through the data structures itself to get the relevant information, as done in the demo code for keyboard.

Data Details Structure (`HID_DATA_DETAILS`)

TABLE 8: DATA DETAILS STRUCTURE (`HID_DATA_DETAILS`)

Data Type	Data Field	Description
BYTE	reportLength	Expected length of the parent report.
BYTE	reportID	First byte of the parent report.
BYTE	bitOffset	Bit offset within the report.
BYTE	bitLength	Length of the data in bits.
BYTE	count	Remaining message after this data.
BYTE	signExtend	Sign-extend the data.

The user application must populate the HID_DATA_DETAILS data structure using the parsed information before querying for data from the input report received from the device (as done in the keyboard demo code). This is passed as an argument to the USBHostHID_ApiImportData() function.

Report Information Data Structure (USB_HID_DEVICE_RPT_INFO)

TABLE 9: REPORT INFORMATION DATA STRUCTURE (USB_HID_DEVICE_RPT_INFO)

Data Type	Data Field	Description
WORD	reportPollingRate	Rate at which the device should be polled for this report.
BOOL	haveDesignatorMax	Flag indicates if the Designator MAX is present in the report.
BOOL	haveDesignatorMin	Flag indicates if the Designator MIN is present in the report.
BOOL	haveStringMax	Flag indicates if the String MAX is present in the report.
BOOL	haveStringMin	Flag indicates if the String MIN is present in the report.
BOOL	haveUsageMax	Flag indicates if the Usage MAX is present in the report.
BOOL	haveUsageMin	Flag indicates if the Usage MIN is present in the report.
WORD	designatorMaximum	Designator MAX value.
WORD	designatorMinimum	Designator MIN value.
WORD	designatorRanges	Range of Usage page.
WORD	designators	Total Number of Designators.
WORD	rangeUsagePage	Range of Usage page.
WORD	stringMaximum	String MAX value.
WORD	stringMinimum	String MIN value.
WORD	stringRanges	Range of String value.
WORD	usageMaximum	MAX usage value.
WORD	usageMinimum	MIN usage value.
WORD	usageRanges	Range of Usages.
BYTE	collectionNesting	Level of collection nesting.
BYTE	collection	Number of collections.
BYTE	designatorItems	Number of designator items.
BYTE	firstUsageItem	First Usage Item.
BYTE	firstDesignatorItem	First Designator Item.
BYTE	firstStringItem	First String item.
BYTE	globalsNesting	Levels of nesting of globals in a report descriptor.
BYTE	maxCollectionNesting	MAX collection of nesting within a report descriptor.
BYTE	maxGlobalNesting	MAX globals nesting within a report descriptor.
BYTE	parent	Current value of parent within a collection.
BYTE	reportItems	Total number of report items.
BYTE	reports	Total number of reports.
BYTE	sibling	Next level of collection nesting.
BYTE	stringItems	Total number of String Items.
BYTE	strings	Total number of Strings.
BYTE	usageItems	Total number of Usage Items.
BYTE	usages	Total number of Usages.
BYTE	interfaceNumber	Interface number for the current report.
HID_GLOBALS	globals	List of globals for the current report. Refer to Table 8 for details of data structure.

Item List Data Structure (`USB_HID_ITEM_LIST`)

TABLE 10: ITEM LIST DATA STRUCTURE (`USB_HID_ITEM_LIST`)

Data Type	Data Field	Description
`HID_COLLECTION*`	collectionList	Array of structures storing collection details.
`HID_DESIGITEM*`	designatorItemList	Array of structures storing designator items.
`HID_GLOBALS*`	globalsStack	Array of structures global details.
`HID_REPORTITEM*`	reportItemList	Array of report items.
`HID_REPORT*`	reportList	Array of report list.
`HID_STRINGITEM*`	stringItemList	Array of string Items.
`HID_USAGEITEM*`	usageItemList	Array of usage Items.
`BYTE*`	collectionStack	Array stores parent index for the collection.

PERFORMING A TRANSFER

Normal communication with the device can be initiated after the device is enumerated and the application is aware of the report format. The HID client is now ready to receive the transfer requests from the application.

The application will require the following interface functions to schedule the transfers:

- `BOOL USBHostHID_ApiDeviceDetect (void);`

This function allows regular monitoring of the status of the device. This function returns TRUE if the HID client is ready to accept the transfer request.

- `BYTE USBHostHID_ApiGetReport(WORD reportid, BYTE interfaceNum, BYTE size,BYTE* data);`

This function is called by the application to receive the input report from the device. Input parameters to this function are known to the application during enumeration as described in the earlier section. This function returns `USB_SUCCESS` if the transfer request is successful.

See the **"Normal Client Driver Operation"** section.

- `BYTE USBHostHID_ApiSendReport(WORD reportid, BYTE interfaceNum, BYTE size,BYTE* data);`

This function is called by the application to send the output report to the device. The input parameters to this function are known to the application during enumeration as described in the earlier section. This function returns `USB_SUCCESS` if the transfer request is successful.

See the **"Normal Client Driver Operation"** section.

- `BOOL USBHostHID_ApiTransferIsComplete (BYTE* errorCodeDriver, BYTE* byteCount);`

This function indicates whether the last transfer is complete. If the function returns TRUE, the returned byte count and error code are valid.

- `BOOL USBHostHID_ApiImportData(BYTE *report, WORD reportLength, HID_USER_DATA_SIZE *buffer, HID_DATA_DETAILS *pDataDetails);`

This function can be used by the application to extract data from the input reports. On receiving the input report from the device, the application can call this function with the required inputs, `HID_DATA_DETAILS`.

EXAMPLE 1: HID DATA TRANSFER FROM THE DEVICE TO THE HOST

```
error = USBHostHID_ApiGetReport (reportid, interfaceNum, size, &data);
if (!error)
{
 while (!USBHostHID_ApiTransferIsComplete (&error, &count))
  {
   USBTasks();
  }
}
```

EXAMPLE 2: HID DATA TRANSFER FROM THE HOST TO THE DEVICE

```
error = USBHostHID_ApiSendReport (reportid, interfaceNum, size, &data);
if (!error)
{
 while (!USBHostHID_ApiTransferIsComplete (&error, &count))
  {
   USBTasks();
  }
}
```

CONCLUSION

The USB Embedded Host HID class provides a simple interface to popular USB Human Interface devices. Embedded applications can now easily take advantage of this flexible HID host in numerous applications involving external inputs to user applications.

RESOURCES

- *AN1140* "*USB Embedded Host Stack*" (http://www.microchip.com/usb)
- *AN1141* "*USB Embedded Host Stack Programmer's Guide*" (http://www.microchip.com/usb)
- Universal Serial Bus web site (http://www.usb.org)
- Microchip Technology Inc. web site (http://www.microchip.com/usb)

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/02/08