Microsoft Corporation

HID Over I^{2}C Protocol Specification

Device Side

Fred Bhesania

(Last reviewed on: February 10, 2016)

Version 1.00

Table of Contents

1 PREFACE

The Human Interface Device protocol was first defined for USB attached input devices (e.g. keyboards, mice, remote controls, buttons, etc.). The protocol itself is bus agnostic and has been ported across other transports, including Bluetooth™ and other wired and wireless technologies.

This specification document will build on the USB defined specification and identify the protocol, procedures and features for simple input devices to talk HID over I^{2}C. This specific document will provide details on the DEVICE side of the protocol. Details on HOST side optimizations are captured in a separate specification.

1.1 VERSION HISTORY

Date	Changes
May 12, 2010	First Draft and overall layout identified
Oct 6, 2010	Added Descriptors Section and necessary data structures
Dec 10, 2010	V0.5 – Draft with initial feedback from team.
Jan 10, 2011	V0.6 – Reviewed and prototyped input reports and descriptors
Jan 27, 2011	V0.7 – Draft ready for internal MSFT review.
Feb 28, 2011	V0.75 – Power, partner feedback, samples added
Mar 18, 2011	V0.8 – Power, Error Handling and internal team review feedback
May 20, 2011	V0.81 – Address partner feedback and protocol diagrams
Oct 31, 2011	V0.9 – Extended HID Descriptor, ACPI changes, example updates.
Jan 30, 2012	V0.91 – Fixed Errors, Get/Set_Report with Write-Read format & Report ID >=15
Mar 21, 2012	V1.0 – Addressed any final feedback and spelling mistakes.

1.2 DEFINITIONS

HID – [Human Interface Device] This term is commonly used to refer to either the protocol or the device itself. In this document, we will use the word “HID” when referring to the device and “HID Protocol” when referring to protocol in definition.

I^{2}C - I^{2}C (Inter-Integrated Circuit) is a multi-master serial single-ended bus that is used to attach low-speed peripherals to a motherboard.

PID – [Physical Interface Device] A special class of HIDs that send physical (tactile) output as well as input along with definition of how the HIDs interact with human hands. This class has not been very popular and has not been supported in many modern operating systems.

SPB – [Simple Peripheral Bus] For the purpose of this document the specific SPB are I^{2}C , SPI, etc. This specification is focused on supporting HID over I^{2}C.

USB – [Universal Serial Bus] Universal Serial Bus (USB) is an industry standard which defines the cables, connectors and protocols used for connection, communication and power supply between USB compliant Hosts and Devices.

HOST – The term HOST refers to the I^{2}C controller or the software on the operating system that governs the operation of the controller and the HID over I^{2}C protocol.

DEVICE – The term DEVICE refers to the I^{2}C peripheral that is connected to the I^{2}C controller and functions in compliance with the HID over I^{2}C protocol specification.

Class Driver – A software driver that is provided in an operating system that is capable of working with different hardware devices that are built in compliance to a class specification.

1.3 DOCUMENT CONVENTIONS

This specification uses the following typographic conventions

Example of Convention	Description
Get_Report, Report	Words in bold with initial letter capitalized indicate elements with special meaning such as requests, descriptors, descriptor sets, classes, or subclasses.
Data, Non-Data	Proper-cased words are used to distinguish types or categories of things. For example Data and Non-Data type Main items.
bValue, bcdName, wOther	Placeholder prefixes such as 'b', 'bcd', and 'w' are used to denote placeholder type. For example: • *b* bits or bytes; dependent on context • *bcd* binary-coded decimal • *d* descriptor • *i* index • *w* word
[bValue]	Items inside square brackets are optional.
…	Ellipses in syntax, code, or samples indicate 'and so on…' where additional optional items may be included (defined by the developer).
{this (0) | that (1)}	Braces and a vertical bar indicate a choice between two or more items or associated values.

`Collection`
`End Collection`
	This font (`Courier`) is used for code, report descriptor, pseudo-code, and samples.

This document is intended to provide a set of general and unambiguous rules for implementing the HID protocol over I^{2}C for a DEVICE, with the goals of hardware software compatibility (including software reuse), performance and power management.

In this specification document, the following symbols are used to identify required and optional features.

Symbol	Description
[M]	Mandatory to support in hardware and software
[O]	Optional to support in hardware and/or in software
[C]	Conditional support (required to fully implement an optional feature)

1.4 RELATED DOCUMENTS

The following are related documents and provide guidance and background on HID.

Specification Name	Specification Location	Notes
HID over USB	http://go.microsoft.com/fwlink/?LinkID=210384	This is the first document that a new reader should review.
HID over Bluetooth	http://go.microsoft.com/fwlink/?LinkID=210385	Provides a mapping of HID to a wireless transport (Bluetooth).
HID Usage Tables	http://go.microsoft.com/fwlink/?LinkID=210386	Provides a summary of the Usage Tables that can be leveraged over any transport.
I^{2}C Specification User Manual	http://www.nxp.com/documents/user_manual/UM10204.pdf	Specification that identifies how the I^{2}C bus works at a protocol level.
ACPI 5.0 Specification and Overview	http://acpi.info/spec.htm	This specification describes the structures & mechanisms to design operating system-directed power management & advanced configuration architectures.

2 INTRODUCTION

This document describes how to use Human Interface Device (HID) class devices over a simple peripheral bus transport, with an immediate focus on I^{2}C. This specification shares some similar concepts with the USB HID specification, but they are not reused and are not intended to be identical. The HID USB or Bluetooth Specifications are recommended pre-reading for understanding the content of this document. See the referenced documents section at the beginning of this document. The HID class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID class devices include:

• Keyboards and pointing devices; for example, standard mouse devices, trackballs, and joysticks

• Front-panel controls; for example, knobs, switches, buttons, and sliders

• Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices; for example, data gloves, steering wheels, phone's keypads and rudder pedals

• Devices that may not require human interaction but provide data in a similar format to HID class devices; for example, bar-code readers, thermometers or other forms of sensors.

The HID protocol was originally targeted at human interface devices; however, HID protocol is very useful for any application that requires low-latency input-output operations to an external interface, and the ability for that device to describe itself. Many typical HID class devices include indicators, specialized displays, audio feedback, and force or tactile feedback.

The HID protocol is an asymmetric protocol that identifies roles for the **HOST** and the **DEVICE**. The protocol will define a format (**Descriptors**) for the DEVICE to describe its capabilities to the HOST. Once the HOST understands the format of communication with the DEVICE, it programs the DEVICE for sending data back to the HOST. The HID protocol also identifies ways of sending data to the DEVICE as well as status checks for identifying the current state of the device.

The remainder of this protocol specification is broken out in to the following parts

- **Scenarios** – A brief description of the potential scenarios and the goals of this specification to address existing problems around these scenarios
- **Descriptors** – A summary of the data elements that will be exchanged between the HOST and the DEVICE to enable enumeration and device identification.
- **Reports** – A summary of data elements that will be exchanged between the HOST and the DEVICE to enable transfer of data in the form of INPUT and OUTPUT reports.
- **Requests** – A summary of the commands and response between the HOST and the DEVICE.
- **Power Management and Error Correction** – A summary on the different commands that are sent from HOST to DEVICE to set/get state information and to address any protocol errors that may occur over the selected transport.

The Appendix section provides examples and for a specific device end to end.

3 SCENARIOS

The following section provides example of user and developer scenarios that are addressed via this protocol specification.

DEVICE Hardware Developer Scenario (Basic HID Touchpad)

The following is a scenario that demonstrates the value of HID over I^{2}C from the perspective of an independent hardware vendor (IHV). Imagine a scenario where a device IHV needs to port their existing HID touchpad from one bus to a different bus. The example is one for a touchpad controller that used to initially be connected over USB (leveraging HID) and now can be easily migrated to I^{2}C, while still leveraging HID and this new standard.

The expected experience and the associated benefits are as follows:

- The HID descriptor and the report descriptor can remain analogous from an application compatibility perspective. HOST software is responsible for maintaining application compatibility. Do note that the HID Descriptor fields in this specification do not need to be identical to the HID Descriptor fields in the USB HID specification or the Bluetooth HID specification.
- A singular driver is needed to channel the HID reports and descriptor sets from I^{2}C and pass the data to higher level software. This driver could be provided by the manufacturer of the chipset or could be a class driver that is written once for an advanced operating system.
- Existing software, that understands HID reports, does not need to be modified to read the HID reports that are sent over I^{2}C

System Integrator Scenario (HID Sensor)

Consider a scenario where a System Integrator needs to select a sensor module for their latest system under development. With the recent updates to HID Usage Tables to support sensors, a sensor manufacturer can now develop a sensor module that can be internally connected to a system over I^{2}C and externally connected to the same system via USB without significant protocol modifications. The integrator can now test sensors from different manufacturers easily and source the best part to meet their requirements.

The expected experience and the associated benefits are as follows:

- Standardized solution for sensors over multiple buses, allowing the integrator to source and test from multiple sources.
- The software driver support for Sensors over HID on the system side can work seamlessly with the internal or the external sensors.
- Generic industry tests are available to the integrator to validate the solutions.
- Generic software and hardware based emulation suites are available based on the standardized protocol (imagine a simple bus analyzer that knows how to parse HID over I^{2}C).

Software Developer Scenario (HID Touch Screen)

Consider a scenario where independent software vendor (ISV) is developing software for a touch screen. With the introduction of HID over I^{2}C, the ISV is capable of leveraging their existing software to work with the new HID I^{2}C touch screen. This solution also allows the operating system to leverage existing software support for touch screens on a new transport.

The expected experience and the associated benefits are as follows:

- Ability to leverage existing software that is HID ready. No special drivers needed to talk a vendor proprietary protocol for I^{2}C touch screens.
- Ability to reuse the ISV's regression tests and validation test environment for HID.
- The software driver support for the touch screen over HID on the HOST can work seamlessly with the internal or the external sensors
- Ability to reuse the ISV's simulation environment for HID.

4 I^2C SPECIFIC DETAILS

This section of the specification identifies details about the transport and how it needs to be configured to support HID over I^2C.

4.1 BUS SPEEDS

This protocol supports the following currently advertised I^2C bus speeds.

- Standard Mode (Sm) – Generally used for lower bandwidth devices up to 100 kbit/s. e.g. keyboard/mouse
- Fast Mode (Fm) – Generally used for lower bandwidth devices up to 400 kbit/s. e.g. Accelerometer
- Fast Mode Plus (Fm+) – Generally used for lower bandwidth devices up to 1 Mbit/s. e.g. complex HID device of Accelerometer , Gyroscope and sensor fusion
- High Speed Mode (Hs-mode) – Generally used for lower bandwidth devices up to 3.4 Mbit/s. e.g. compound sensors like multi-axis Accelerometers and Gyroscopes on a single device.

The DEVICE and the HOST must both support the necessary speed of communication as identified in the ACPI Specific Details section. It is up to the HOST to preselect the bus speed during initialization.

It is also possible for a system to have a combination of Hs and Fm devices connected to the same Master. This configuration would require an *interconnection bridge* to have the different bit rates between the different devices.

4.2 SCHEMATIC LAYOUT

The following is an example block diagram of the HOST-DEVICE interface.

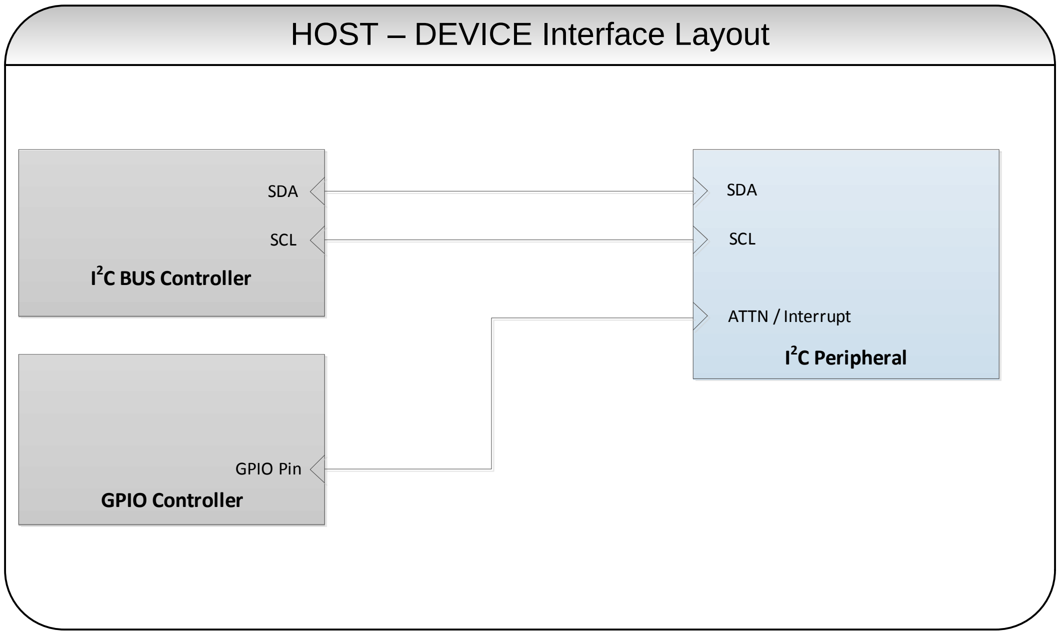

Figure 1: HOST-DEVICE Interface Layout

Notes

- HID I^{2}C peripheral (DEVICE) must leverage the regular SDA and SCL lines and provide a mandatory interrupt to indicate data and status.
- HID I^{2}C specification allows multiple DEVICEs (I^{2}C peripherals) to be connected to a single HOST (I^{2}C Bus Controller), but each DEVICE must have its dedicated Interrupt line and I^{2}C address.
- Multi-master configurations are not supported as part of this specification.
- Additional details around the mechanical requirements are beyond the scope of this specification.
- Electrical requirements are beyond the scope of this specification. The HOST and the DEVICE controller shall be in compliance with the I^{2}C specification.
- The schematic above shows an interrupt layout. Interrupt line configuration may vary and are outside the scope of this specification. The interrupt may be both a function and wakeable interrupt.

4.3 BYTE ORDERING

All multiple byte fields (e.g. in standard descriptors, requests, responses, etc) are interpreted in little-endian notation. Multiple byte values are sent out onto the bus least-significant byte (LSB) first, followed by the next LSB, through to the most significant byte (MSB) last.

4.4 OVERALL HID I²C EFFICIENCY

The table below identifies the maximum size of a report (based on available bus bandwidth) that a HID I²C device can send to the HOST for each of the supported speeds. Each cell in the table is calculated as follows

$$Max\ Report\ Size = \frac{Bus\ Speed}{Report\ Rate}$$

It is an exercise to the system integrator to identify and program the correct I²C Bus Speed based on the HID device(s) connected to that I²C controller. Do note that the table below is a theoretical maximum and a system implementer should anticipate about 50% of the values below. This table is meant to be a high level description of the perceived report sizes and not meant to be an accurate value.

Rate of Sending Reports	BusSpeed_Sm (100KHz)	BusSpeed_Fm (400KHz)	BusSpeed_Fm+ (1MHz)	BusSpeed_Hs (3.4 MHz)
1KHz (1ms)	10Bytes	40Bytes	100Bytes	300Bytes
100Hz (10ms)	100Bytes	400Bytes	1KBytes	3KBytes
10Hz (100ms)	1KBytes	4Kbytes	10KByte	30KBytes
1Hz (1sec)	10KBytes	40Kbytes	100KByte	Beyond supported report size

The section below provides a more accurate calculation of HID I²C efficiency for the most common case, the input report. For every input report, there are 29 bits of packet overhead (calculation shown below) based on a 7bit addressing scheme.

Read

1 Start bit
7 Address bits*
1 R/W bit
1 Ack bit
16 Data bits (Length)
2 Ack bits
1 STOP bit

Total = 29 bits
*For 10 bit addressing, please use 32.

For every REPORT of data there are 8 bits data + 1 bit ACK. Hence the overall efficiency of the hid protocol over I^2C is as follows:

Payload to transfer in Bytes (n) Bytes	**Total bits to Transmit (9*n + 29 Bits)**	**Actual Throughput % (Payload bits / Total Transmitted bits)**	**Latency on 400KHz in mSec**	**Latency on 1MHz in mSec**
1	38	24%	0.09	0.03
5	74	61%	0.18	0.07
10	119	76%	0.29	0.11
20	209	86%	0.51	0.21
50	479	94%	1.7	0.47

5 DESCRIPTORS

The following section identifies the key data structures (referred to as descriptors) that need to be exchanged between the HOST and the DEVICE during the startup phase and the data delivery phase.

5.1 HID DESCRIPTOR

The HID Descriptor is the top-level mandatory descriptor that every I^{2}C based HID DEVICE must have. The purpose of the HID Descriptor is to share key attributes of the DEVICE with the HOST. These attributes accurately describe the version that the protocol is compliant towards as well as additional data fields of the device.

5.1.1 HID DESCRIPTOR FORMAT

The device must expose a HID Descriptor with the following fields.

Table 1: HID Descriptor Layout

Byte Offset	Field	Size (Bytes)	Type	Description
0	**wHIDDescLength**	2	WORD	The length, in unsigned bytes, of the complete Hid Descriptor
2	**bcdVersion**	2	BCD	The version number, in binary coded decimal (BCD) format. DEVICE should default to 0x0100
4	**wReportDescLength**	2	WORD	The length, in unsigned bytes, of the Report Descriptor.
6	**wReportDescRegister**	2	WORD	The register index containing the Report Descriptor on the DEVICE.
8	**wInputRegister**	2	WORD	This field identifies, in unsigned bytes, the register number to read the input report from the DEVICE.
10	**wMaxInputLength**	2	WORD	This field identifies in unsigned bytes the length of the largest Input Report to be read from the Input Register (Complex HID Devices will need various sized reports).
12	**wOutputRegister**	2	WORD	This field identifies, in unsigned bytes, the register number to send the output report to the DEVICE.
14	**wMaxOutputLength**	2	WORD	This field identifies in unsigned bytes the length of the largest output Report to be sent to the Output Register (Complex HID Devices will need various sized reports).
16	**wCommandRegister**	2	WORD	This field identifies, in unsigned bytes, the register number to send command requests to

				the DEVICE
18	**wDataRegister**	2	WORD	This field identifies in unsigned bytes the register number to exchange data with the Command Request
20	**wVendorID**	2	WORD	This field identifies the DEVICE manufacturers Vendor ID. Must be non-zero.
22	**wProductID**	2	WORD	This field identifies the DEVICE's unique model / Product ID.
24	**wVersionID**	2	WORD	This field identifies the DEVICE's firmware revision number.
26	**RESERVED**	4	RESERVED	This field is reserved and should be set to 0.

The HID Descriptor format in this specification is different from the HID descriptor in other transport specifications (e.g. USB). Only the critical common fields for the HID descriptor across transports are captured in the table. It is the responsibility of the HID HOST software to identify and fill in the rest of the fields leveraging OS-specific or ACPI specific data structures.

wHIDDescLength

This mandatory field contains the length (in Unsigned Bytes) of the entire HID Descriptor. The HOST uses this value to validate the size of the HID Descriptor in Bytes and to retrieve the complete descriptor. It is fixed to 0x1E.

bcdVersion

This mandatory field contains the version number of the HID over I^{2}C protocol Specification that the device is compatible with. For version 1.00, **bcdVersion** shall be set to 0x0100.

wReportDescLength

The mandatory field contains the length (in Unsigned Bytes) of the Report Descriptor. The maximum size of the report descriptor is 2^16-1 bytes (64KB). For additional details on Report Descriptors, please review section 5.2.

wReportDescRegister

The mandatory field contains the register index that is used to retrieve the Report Descriptor. The register index must be readable at any time and cannot be modified. The register field must not contain 0 as the register value.

For additional details on Report Descriptors, please review section 5.2.

wInputRegister

The mandatory field contains the register index that is used to retrieve a report from the DEVICE to the HOST. The register must contain valid data as soon as the interrupt line is asserted by the DEVICE and will be read by the HOST. The register field is always necessary (even if the device does not expose an input report), and must always contain a valid register number (i.e. non-zero value).

For additional details on the usage of this register, please review section 5.2.

wMaxInputLength

The mandatory field contains the maximum length (in unsigned bytes) of an Input Report. This length field should also include the 2 Byte length field and the optional report ID. This length field should be identical to the largest Input Report's length as identified in the Report Descriptor. If a device supports multiple Top Level Collections, the value stored in the wMaxInputLength must be the length of the longest report. The maximum size of an input report is 2^16-4 Bytes (details in the report section). Many modern HOSTS recommend that Input Report Length be shorter than 63 Bytes for each report (longer reports should be separated via Report IDs). Please see the Sizes and Constants section for more information.

If a report descriptor does not identify any input reports, this field must contain 2.

For additional details on Report Descriptors, please review section 5.2 and the example in the appendix.

wOutputRegister

The mandatory field contains the register index that is used to send a report from the HOST and the DEVICE. The register field should contain 0 if there is no output report(s) defined in the report descriptor; else it should contain a valid register number.

For additional details on the usage of this register, please review section 5.2.

wMaxOutputLength

The mandatory field contains the maximum length (in unsigned bytes) of an Output Report. This length field should be identical to the largest output report's length as identified in the Report Descriptor. The maximum size of an output report is 2^16-4 Bytes (64KB). If a report descriptor does not identify any output reports, this field must contain 0.

For additional details on Report Descriptors, please review section 5.2.

wCommandRegister

The mandatory field contains the register index that is used to send *COMMAND Request* between the HOST and the DEVICE. The register field must not contain 0 as the register value.

For additional details on the usage of this register, please review section 5.2.

wDataRegister

The mandatory field contains the register index that is used to exchange data (as a response to the *COMMAND Request*) report between the HOST and the DEVICE. The register field cannot contain 0 as the register value.

For additional details on the usage of this register, please review section 5.2.

wVendorID

The mandatory field contains the unique Vendor ID of the DEVICE. The value is a 2-Byte ID and is used to uniquely identify the manufacturer of the DEVICE silicon. The value should be non-zero.

wProductID

The mandatory field contains the unique Device ID of the DEVICE. The value is a 2-Byte ID and is used to uniquely identify the model of the DEVICE silicon. The Device ID in conjunction with the Vendor ID helps provide a unique identifier for the specific device.

wVersionID

The mandatory field contains the unique Revision value to identify the version of firmware running on the DEVICE. The value is a 2-Byte BCD ID and is up to the vendor's discretion. It is highly recommended to increment the wVersionID when the firmware on the DEVICE is updated.

RESERVED

The mandatory section is reserved for future proofing the specification. It should be set to 0.

Notes

1. The following register IDs must all be unique: *wReportDescRegister; wInputRegister; wOutputRegister; wCommandRegister; wDataRegister.*
2. The *wVendorID, wProductID, wVersionID* fields may not be used to generate the device's hardware ID, if it is generated from the _HID field from ACPI.

5.1.2 HID DESCRIPTOR RETRIEVAL

The HID Descriptor must be stored on the DEVICE firmware where it can be retrieved on DEVICE initialization. During DEVICE initialization, the HOST must retrieve the HID Descriptor. To retrieve the HID Descriptor, the HOST will read the HID Descriptor Register. The HID Descriptor Register is an implementation detail that is specified by the device manufacturer and is device specific and is described in the ACPI Specification section later in this specification. The HID Descriptor Register is stored in ACPI and will be discussed in Section 10.

There are a variety of ways a HOST may choose to retrieve the HID Descriptor from the DEVICE. The following is a preferred implementation but should not be considered the only implementation. A HOST may read the entire HID Descriptor in a single read by issuing a read for 30 Bytes to get the entire HID Descriptor from the DEVICE. However, the HOST is responsible for validating that

1. The BCDVersion is V01.00 (later revisions may have different descriptor lengths), and
2. The value stored in *wHIDDescLength* is 30 (Bytes) for V1.00 descriptors.

Alternatively, another method may also be possible though less efficient. A HOST may issue a read for the first 4 Bytes of the HID descriptor register. This will provide the HOST the version number and length of the HID Descriptor. Once the HOST knows the length and the version number, the HOST should issue a second read request to the DEVICE with the size field matching the correct size identified in the HID Descriptor. The DEVICE at that time will return the full HID descriptor.

The HOST is allowed to request the HID Descriptor again from the device using the method above without resetting the DEVICE. The DEVICE must follow the same procedure above and return the full HID Descriptor at any time during its operation.

The following is an illustration of the protocol between the HOST and the DEVICE to retrieve the HID Descriptor.

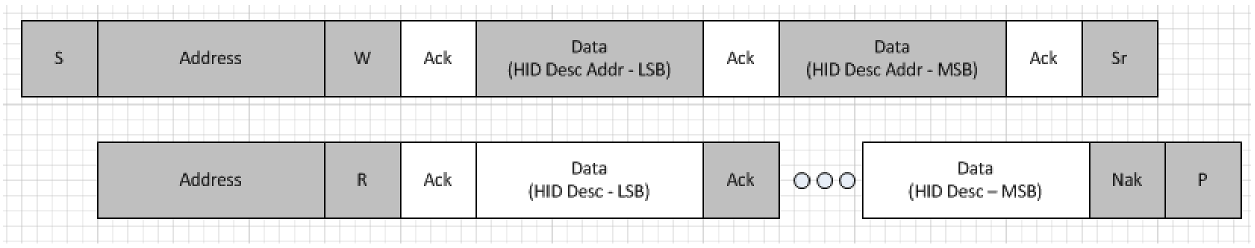

Figure 2: HID Descriptor Retrieval

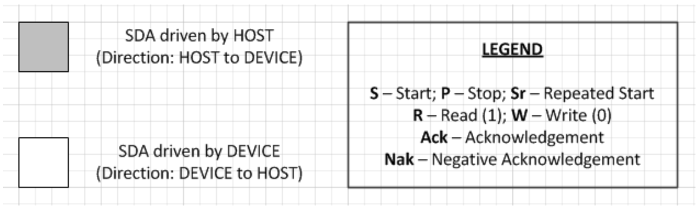
SDA driven by HOST (Direction: HOST to DEVICE)
SDA driven by DEVICE (Direction: DEVICE to HOST)
LEGEND
S – Start; P – Stop; Sr – Repeated Start
R – Read (1); W – Write (0)
Ack – Acknowledgement
Nak – Negative Acknowledgement

Figure 3: Legend

5.2 REPORT DESCRIPTOR

A Report Descriptor is composed of report items that define one or more top-level collections. Each top-level collection defines one or more HID reports. The Report Descriptor is a mandatory descriptor that is stored in the HID DEVICE and is retrieved by the HOST on initialization. The concept of a Report Descriptor for HID over I^{2}C is identical to the notion of a Report Descriptor in the original USB HID Class Specification V1.11 or later.

5.2.1 REPORT DESCRIPTOR FORMAT

The Report descriptor is unlike other descriptors in that it is not simply a table of values. The length and content of a Report descriptor vary depending on the number of data fields required for the device's report(s). The Report descriptor is made up of items that provide information about the device. For more details on report descriptors, please review the HID over USB Specification.

The following descriptors are outside of the scope of this specification and are not supported:

- Physical Descriptor
- Vendor Specific Descriptor

Vendors wishing to add these or other proprietary descriptors need to expose a secondary I²C device.

5.2.2 REPORT DESCRIPTOR RETRIEVAL

A single Report Descriptor must be stored on the DEVICE. During DEVICE initialization, the HOST gets the location and length of the report descriptor from **wReportDescRegister and wReportDescLength** in the HID Descriptor.

To retrieve the Report Descriptor from the device, the HOST will read the register identified in **wReportDescRegister** for a length of **wReportDescLength.**

If the HOST driver is unable to retrieve the entire report descriptor or if there are errors, a HOST should reset/reinitialize the DEVICE and start the entire enumeration sequence again.

If the DEVICE provides a malformed report descriptor, the behavior of the HOST is operating system defined and may vary from one operating system to another. It is not necessary for the HOST to re-retrieve a report descriptor.

In general, the HOST software/driver will first retrieve the report descriptor before reading the reports from the device. However, it is acceptable for a HOST to retrieve the reports from a DEVICE without retrieving the Report Descriptor. This may be the case for closed configuration platforms or vertical systems where the format of the reports is known to the HOST at system initialization.

The following is an illustration of the protocol between the HOST and the DEVICE to retrieve the Report Descriptor.

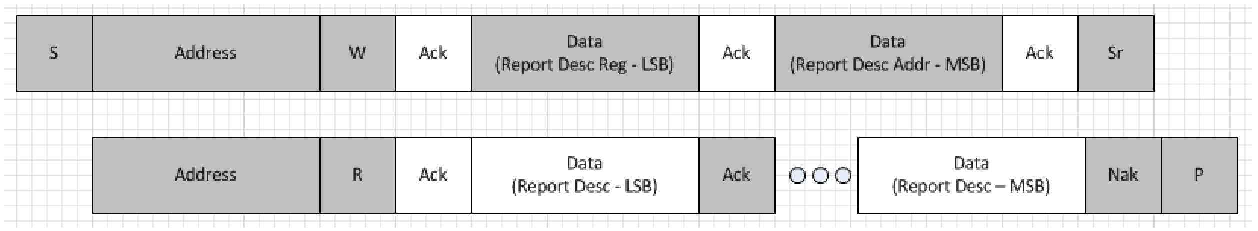

Figure 4: Report Descriptor Retrieval

6 REPORT PROTOCOL

The Report is the fundamental data element that is exchanged between the HOST and the DEVICE. The definition of reports will be synonymous to their definition and naming convention in the USB HID Class Specification to prevent any confusions.

The following convention is used for reports

- **Input Reports** – The uni-directional Report that is sent from the DEVICE to HOST
- **Output Reports** – The uni-directional Report that is sent from the HOST to the DEVICE
- **Feature Report** – A bi-directional Report that can be sent from the HOST to the DEVICE or from the DEVICE to the HOST.

The sections below will elaborate on each of these report types. While this specification does allow for longer reports, the original HID specification curtailed the size of a single report to be shorter than 63 Bytes. Longer reports can be constructed using Report IDs or different Top Level Collections in HID.

6.1 INPUT REPORTS

The input reports are generated on the DEVICE and are meant for communication in the direction of DEVICE to HOST over the I^{2}C transport. When the DEVICE has active data it wishes to report to the HOST, it will assert the Interrupt line associated with the HID protocol on the DEVICE. When the HOST receives the Interrupt, it is responsible for reading the data of the DEVICE via the Input Register (field: **wInputRegister**) as defined in the HID Descriptor. The HOST does this by issuing an I^{2}C read request to the DEVICE.

It is the responsibility of the DEVICE to assert the interrupt until all the data has been read for that specific report. After reading the Input Report, the DEVICE can continue to or reassert the interrupt if there are additional Input Report(s) to be retrieved from the DEVICE.

6.1.1 SINGLE TOP LEVEL COLLECTIONS

The length of the read is identified in the HID Descriptor (field: **wMaxInputLength**). This field identifies the number of bytes to read from the input register. Once the entire report has been read by the HOST, the DEVICE is allowed to trigger the next read by asserting the interrupt line. Do note that a device with a single TLC may be optimized and may not provide a Report ID (only mandatory for a device with multiple TLC).

Length (2 Unsigned Bytes)	Report

The value stored in the “Length” field is calculated as follows:

$$\begin{matrix}\textit{Value in}\\ \textbf{Length}\ \textit{field}\end{matrix} = 2Bytes\begin{pmatrix}\#Bytes\ for\ the\\ "\textbf{Length}"\ field\end{pmatrix} + xBytes\begin{pmatrix}\#Bytes\ for\ the\\ "\textbf{Report}"\ Field\end{pmatrix}$$

*Length field should always be greater than 2Bytes for a valid report.

Though it may seem that the length field is redundant, a valid length field is paramount for identifying errors in transmission as well as identifying reset conditions.

6.1.2 MULTIPLE TOP LEVEL COLLECTIONS

DEVICEs with multiple TLCs are quite common and use multiple TLCs to expose different classes of devices embedded in a single Peripheral Controller chip (e.g. keyboard buttons as well as consumer control buttons). If a device has multiple Top Level Collections (TLCs) identified in its report descriptor, then it needs to follow an enhanced protocol.

Every input report must start with a length field (2 bytes), followed by a mandatory report ID (1 byte), and finally followed by the report itself. The DEVICE should send the reports to the HOST in the order that they are generated/updated.

Length (2 Unsigned Bytes)	Report ID (1 Byte)	Report

The value stored in the “Length” field is calculated as follows:

$$\begin{matrix}\textit{Value in}\\ \textbf{Length}\ \textit{field}\end{matrix} = 2Bytes\begin{pmatrix}\#Bytes\ for\ the\\ "\textbf{Length}"\ field\end{pmatrix} + 1Bytes\begin{pmatrix}\#Bytes\ for\ the\\ "\textbf{Report ID}"\ Field\end{pmatrix} + xBytes\begin{pmatrix}\#Bytes\ for\ the\\ "\textbf{Report}"\ Field\end{pmatrix}$$

*Length field should always be greater than 3 Bytes for a valid report.

6.1.3 RETRIEVAL OF INPUT REPORTS

The input report is one of the most popular forms of communication between the DEVICE and HOST. The following sequence of operations explains the process of retrieval of the Input Report.

Sequence	HOST Side	DEVICE Side
Step 1		DEVICE asserts the Interrupt indicating that it has an Input Report to send to HOST
Step 2	HOST issues a READ request over the I^{2}C protocol, after it receives the interrupt	
Step 3		DEVICE returns the length (2 Bytes) and the entire Input Report.
Step 4		If the DEVICE has no more Input Reports to send, it de-asserts the interrupt line.

The protocol trace diagram below provides a summary of how the Input Report is retrieved over I^{2}C.

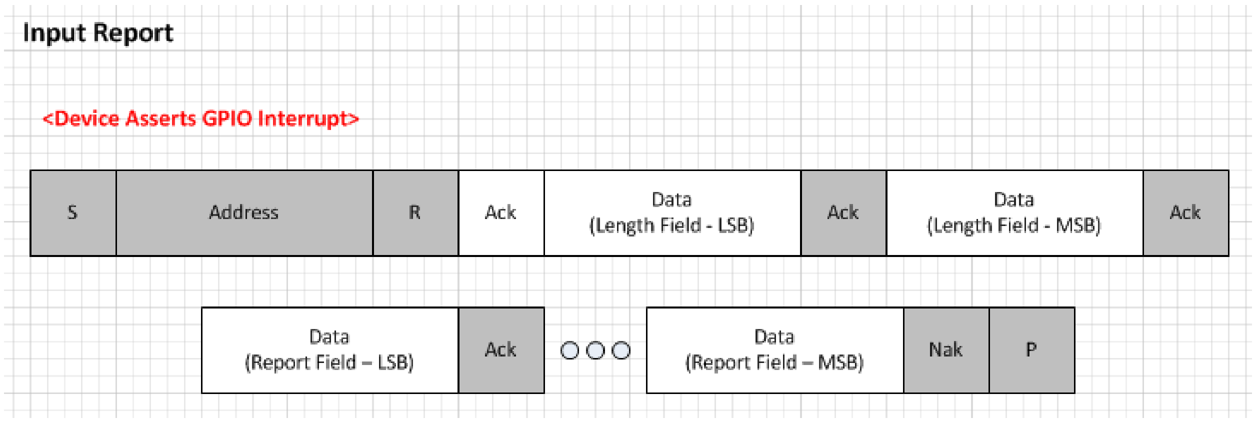
Input Report
<Device Asserts GPIO Interrupt>
S
Address
R
Ack
Data (Length Field - LSB)
Ack
Data (Length Field - MSB)
Ack
Data (Report Field – LSB)
Ack
Data (Report Field – MSB)
Nak
P

Figure 5: Input Report Retrieval

6.2 OUTPUT REPORTS

The output report is generated on the HOST and is meant for communication in the direction of HOST to DEVICE over the I^{2}C transport. When the HOST has active data it wishes to report to the DEVICE, it will write the output report to the output register (field: **wOutputRegister)**.

6.2.1 SINGLE TOP LEVEL COLLECTIONS

The maximum length of the write to the output register is identified in the HID Descriptor (field: **wMaxOutputLength**). This field identifies the maximum number of bytes a HOST is allowed to write to the output register. The payload should look at follows.

The length of the write is identified in the HID Descriptor (field: **wMaxOutputLength**). This field identifies the number of bytes to write to the output register. An Interrupt is NOT asserted for a write operation. Do note that a device with a single TLC may be optimized and may choose to not provide a Report ID (Report IDs are only mandatory for a device with multiple TLC).

Length (2 Unsigned Bytes)	**Report**

The value stored in the "Length" field is calculated as follows:

$$\begin{matrix}\mathit{Value\ in} \\ \textbf{Length}\ \mathit{field}\end{matrix} = 2Bytes\begin{pmatrix}\#Bytes\ for\ the \\ "\textbf{Length}"\ field\end{pmatrix} + xBytes\begin{pmatrix}\#Bytes\ for\ the \\ "\textbf{Report}"\ Field\end{pmatrix}$$

*Length field should always be greater than 2 Bytes for a valid report.

6.2.2 MULTIPLE TOP LEVEL COLLECTIONS

Devices with multiple TLCs are quite common and use multiple TLCs to expose different classes of devices embedded in a single Peripheral Controller chip (e.g. keyboard buttons as well as consumer control buttons). If a device has multiple Top Level Collections (TLCs) identified in its report descriptor, then it needs to follow an enhanced protocol.

Every output report must start with a length field (2 bytes), followed by a mandatory report ID (1 byte), and finally followed by the report itself.

Length (2 Unsigned Bytes)	**Report ID** (1 Byte)	**Report**

The value stored in the "Length" field is calculated as follows:

$$\begin{matrix}\mathit{Value\ in} \\ \textbf{Length}\ \mathit{field}\end{matrix} = 2Bytes\begin{pmatrix}\#Bytes\ for\ the \\ "\textbf{Length}"\ field\end{pmatrix} + 1Bytes\begin{pmatrix}\#Bytes\ for\ the \\ "\textbf{Report ID}"\ Field\end{pmatrix} + xBytes\begin{pmatrix}\#Bytes\ for\ the \\ "\textbf{Report}"\ Field\end{pmatrix}$$

*Length field should always be greater than 3 Bytes for a valid report.

6.2.3 SENDING OUTPUT REPORTS

The output report is one of the less popular forms of communication between the HOST and DEVICE. The following sequence of operations explains the process of sending the Output Report the DEVICE.

Sequence	HOST Side	DEVICE Side
Step 1	HOST issues a WRITE request over the I^{2}C protocol.	
Step 2		DEVICE acknowledges each byte of the report in the I^{2}C write transaction.

The protocol trace diagram below provides a summary of how the Output Report is retrieved over I^{2}C.

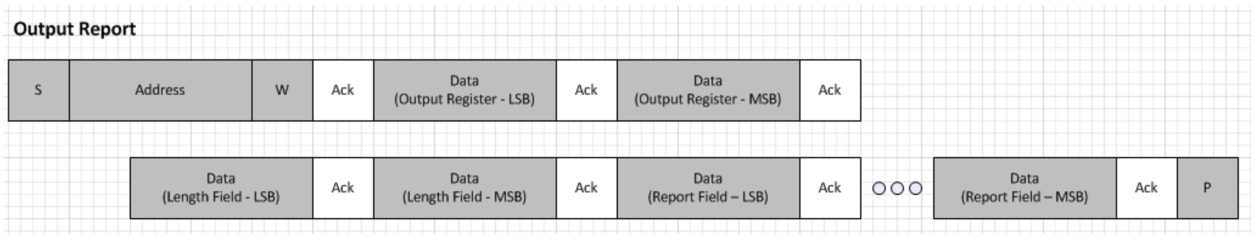
Output Report
S
Address
W
Ack
Data (Output Register - LSB)
Ack
Data (Output Register - MSB)
Ack
Data (Length Field - LSB)
Ack
Data (Length Field - MSB)
Ack
Data (Report Field – LSB)
Ack
Data (Report Field – MSB)
Ack
P

Figure 6: Sending an Output Report

6.3 FEATURE REPORTS

The feature report is a bidirectional report and can be exchanged between the HOST and the DEVICE. They are primarily used by the HOST to program the DEVICE into specific states. In the case of the accelerometer (see Example: Accelerometer Sensor) , the Features are used to set properties like minimal report internal and change sensitivity settings for the accelerometer.

For the HOST to get/set a FEATURE REPORT on the DEVICE, it must use the GET_REPORT and SET_REPORT requests identified in section “Class Specific Requests”.

Unlike Input Reports, changes to the Feature Reports on the DEVICE do not assert interrupts.

Note that it is possible to construct a report descriptor that has multiple reports of one type (feature/input/output) within the same TLC.

6.3.1 GETTING AND SETTING FEATURE REPORTS

For the HOST to get/set a FEATURE REPORT on the DEVICE, it must use the GET_REPORT and SET_REPORT requests. For additional details on GET/SET_REPORT, please refer to the Class Specific Requests Section.

7 REQUESTS

The HID class uses standard commands to manage the device and to also get the current state of the device. All transactions are performed over the Command and the Data Registers. The following section provides details on registers and the requests.

7.1 REGISTERS

The HID device shall expose the following registers.

- **Command Register** – A register that will contain the specific details to issue commands to the DEVICE.
- **Data Register** – A register that will contain the data associated with a command (or corresponding response) that is exchanged between HOST and DEVICE in conjunction with a Command. The data in this register must start with a 2 byte length field.

7.1.1 COMMAND REGISTER

The command register contains a 2 Byte value that will identify the command that is being sent by the HOST to the DEVICE. The Command Register can only be written to by the HOST and read by the DEVICE. In addition to the command, the HOST may also associated Data (either sent to the DEVICE with the command or retrieved from the DEVICE as a response to the command).

Data sent to the command register contains a 2 Bytes value (a High Byte and a Low Byte).

Most Significant Byte (High Byte)

Reserved (4 bits)				Op Code (4 bits)			
7	6	5	4	3	2	1	0

The HIGH Byte must always contain the Operational Code (OPCODE) that is used to uniquely identify the command being issues to the DEVICE.

Least Significant Byte (Low Byte)

Reserved (2 bits)		Report Type (2 bits)		Report ID (4 bits)			
7	6	5	4	3	2	1	0

The LOW Byte generally contains the Report Type and Report ID. It is acceptable for the LOW Byte to contain other values on specific commands (defined by the command Op Code). Details for these will be identified in the Class Specific Commands Section that follows.

Sections of the command register are marked as RESERVED for future use. These RESERVED values must not be used by HOSTs and enforced to 0.

Report Type:

The Report Type should be isolated to one of the values below.

- 00b – Reserved
- 01b – Input
- 10b – Output
- 11b – Feature

The following rules dictate the usage of the report type

- Reserved values shall not be used by the HOST, for cases that need a valid Report Type.
- The HOST shall use the Report Types that are defined in the Report Descriptor for the specific DEVICE.
- If a HOST issues a report type that is not defined for the specific DEVICE (in the report descriptor), the DEVICE shall return 0 in the Data register.

Report ID

The report ID corresponds to the Top Level Collection that the command acts upon. This field should only be used complex devices with multiple input, output or feature report collections. If the device is not a complex HID device, this field must be 0.

7.1.2 DATA REGISTER

The Data register is closely associated with the Command Register. It is used to store the data (and length) of the information associated with the request set in the command register. The Data Register can be filled by the HOST or the DEVICE but must be in accordance with the rules identified for that specific request.

The first 2 bytes of the Data Register will contain the length of the data (i.e. report) being shared in the Data Register. The total number of bytes represented in the first 2 bytes is calculated by appending 2 Bytes to the length of the report being exchanged in the Data Register.

Initial Value

The register must contain 0x00, if read by the HOST, once the device is reset and initialized.

7.2 CLASS SPECIFIC REQUESTS

The following section identifies the class specific requests that are defined in the specification. The table below identifies the requests along with a summary of direction of request and whether it is mandatory.

Request Description (Op Code – 4 bits)	Request Name	Mandatory on HOST? [M] / [O]	Mandatory on DEVICE? [M] / [O]	Description Details
0000b	RESERVED	N/A	N/A	RESERVED
0001b	RESET	M	M	Reset the device at any time
0010b	GET_REPORT	M	M	Request from HOST to DEVICE to retrieve a report (input/feature)
0011b	SET_REPORT	M	M	Request from HOST to DEVICE to set a report (output/feature)
0100b	GET_IDLE	O	O	Request from HOST to DEVICE to retrieve the current idle rate for a particular TLC. This command is not used on modern HOSTS.
0101b	SET_IDLE	O	O	Request from HOST to DEVICE to set the current idle rate for a particular TLC. This command is not used on modern HOSTS.
0110b	GET_PROTOCOL	O	O	Request from HOST to DEVICE to retrieve the protocol mode the device is operating in. This command is not used on modern HOSTS.
0111b	SET_PROTOCOL	O	O	Request from HOST to DEVICE to set the protocol mode the device should be operating in. This command is not used on modern HOSTS.
1000b	SET_POWER	O	M	Request from HOST to DEVICE to indicate preferred power setting
1001-1101b	RESERVED	N/A	N/A	RESERVED
1110b	VENDOR RESERVED	N/A	N/A	This value is reserved for vendor specific extensibility and not used in this specification.
1111b	RESERVED	N/A	N/A	RESERVED

7.2.1 RESET

The following section identifies 2 forms of RESETs:

- HOST Initiated Reset (HIR)
- DEVICE Initiated Reset (DIR)

Host Initiated Reset (HIR)

The reset command is a mandatory request that the HOST can issue to the DEVICE at any time. A HOST shall initially issue a reset to the device at startup to initialize the device.

7.2.1.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	0001b	This value is reserved for the RESET Command
Low Byte		
RESERVED	00b	This value is reserved and must be set to 00b
Report Type	00b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.
Report ID	0000b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.

7.2.1.2 RESPONSE

After the HOST sends the RESET command to the Command Register, the DEVICE shall RESET itself back to the initialized state. At the end of the reset, the DEVICE must also write a 2 Byte value to the Input Register with the sentinel value of 0x0000 (2 Bytes containing 0) and must assert the Interrupt to indicate that it has been initialized (additional details on interrupts is provided in Section 7.4). This is a sentinel value and is interpreted by the HOST as the device has been reset. All registers must contain the correct values upon initialization and the DEVICE must be in ON Power State.

A DEVICE must respond to the RESET Command in less than 5 seconds. If the DEVICE doesn’t assert the interrupt in that timeframe, the HOST may ignore this device and identify to the user as inoperable. A HOST is not required to try consecutive resets to the DEVICE to recover it from this state.

7.2.1.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST sends RESET to Command Register	
Step 2		DEVICE reinitializes itself completely, updates the input register data length field with 0x0000 and then triggers the interrupt.

7.2.1.4 NOTES

- This is a mandatory request and shall be supported on both the HOST end and the DEVICE end.
- The HOST shall issue a RESET request to the DEVICE during first initialization to prepare the device for usage.
- The HOST may choose to RESET the device at any point in time. Upon Reset, the DEVICE should discard all previous states and initialize itself to start from scratch.
- If the DEVICE connects to the HOST over multiple transports, the DEVICE is only responsible for resetting the HID over I^{2}C portion of the interface.
- If the HOST issues an HIR but receives an INPUT Report before it receives an acknowledgment to the HIR, the HOST is recommended to discard that Input Report.

The following is an illustration of the protocol between the HOST and the DEVICE to issue an HIR.

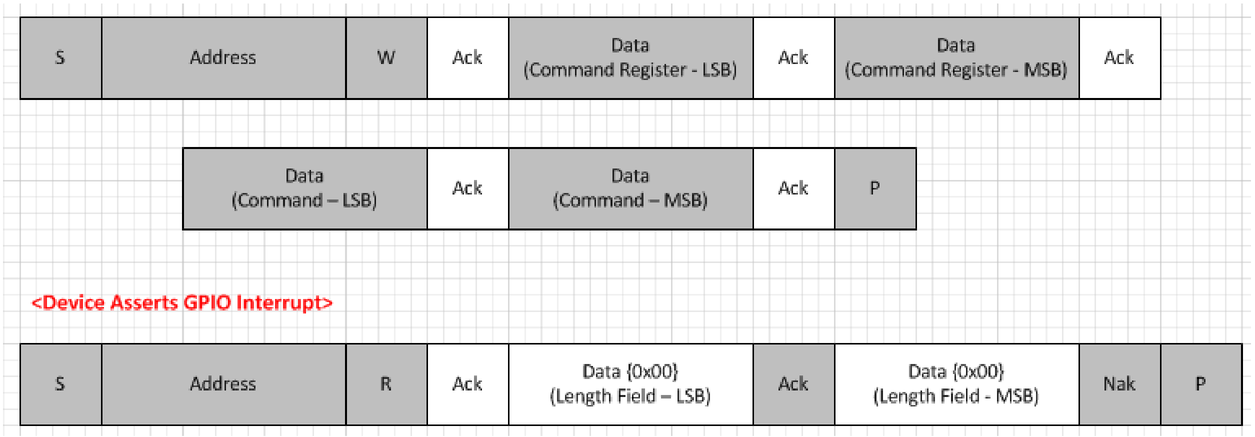
S
Address
W
Ack
Data (Command Register - LSB)
Ack
Data (Command Register - MSB)
Ack
Data (Command – LSB)
Ack
Data (Command – MSB)
Ack
P
<Device Asserts GPIO Interrupt>
S
Address
R
Ack
Data {0x00} (Length Field – LSB)
Ack
Data {0x00} (Length Field - MSB)
Nak
P

Figure 7: Host Initiated Reset Protocol Diagram

<u>DEVICE Initiated Reset (DIR)</u>

There may be operations where the DEVICE needs to reset itself without a request from the HOST. Some examples of the conditions that may trigger this are listed below:

- Electro static discharge
- Error handling mechanism in state machine

- Fault tolerance

The DEVICE is allowed to perform a DEVICE Initiated Reset at any time, as long as it meets the following:

1. The DEVICE must inform the HOST after a DIR that the DEVICE performed a reset and that it is operational again. The mechanism for informing the HOST of a DIR is the same as that for a HIR, which entails that the DEVICE reinitializes itself completely, updates the input register data with 0x0000 and then asserts the interrupt.
2. The DEVICE must finish a DIR without exceeding the specification defined timeouts (see the Error Handling Section).

7.2.2 GET_REPORT

The GET_REPORT command is a mandatory request (if the DEVICE contains Input or Feature reports) that the HOST can issue to the DEVICE at any time after initialization to get a singular report from the DEVICE. The DEVICE is responsible for responding with the last known input or feature for the specified TLC. If the value has not been set for that report yet, the DEVICE must return 0 for the length of that TLCs' report item.

GET_REPORT is often used by applications on startup to retrieve the "current state" of the device rather than waiting for the device to generate the next Input/Feature Report.

7.2.2.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	0010b	This value is reserved for the GET_REPORT Command
Low Byte		
RESERVED	00b	This value is reserved and must be set to 00b
Report Type	{Input (01) | Feature (11)}	The HOST shall always set this to the specified value based on the report type the HOST is trying to get. The DEVICE shall honor this value and return data only if supported in the Report Descriptor for the specified report.
Report ID	xxxxb	The HOST shall always set this to the specified value based on the TLC specific report the HOST is trying to get. The DEVICE shall honor this value per the rules outlined in the notes section below
Third Byte (Optional)		
Report ID	xxxxxxxxb	If the Report ID in the Low Byte is set to the sentinel value, then this optional Third Byte represents the Report ID. Please refer to Notes for details.

*all values ending with **'b'** indicate 'bits'

7.2.2.2 RESPONSE

After the HOST sends the GET_REPORT command to the Command Register, the DEVICE must fill the report in to the DATA Register. A DEVICE may optionally stretch the clock for the subsequent HOST read per repeated start. The Data shall be packaged as follows

- Length of Report (2 Bytes)
- Report including Report ID if defined in Report Descriptor (length identified above)

The following paragraph provides an example. If the length of a mouse report is 4 bytes, then the Data Register should contain the length as 0x0006 followed by 4 bytes of the mouse device's input report. If a

HOST inadvertently tries to read additional data beyond the end of a report, the DEVICE may return arbitrary data.

A DEVICE should not receive a new command from the HOST (except the RESET Command) until this command has been completed and data has been sent to the HOST. If the DEVICE fails to respond back within a HOST defined period of time (generally longer than 5 seconds) the HOST is allowed to RESET the DEVICE.

If the DEVICE receives a request from a HOST to an invalid Report ID, the DEVICE must respond with a Report of ZERO Length.

7.2.2.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST sends GET_REPORT to Command Register	
Step 2		DEVICE fills the Data Register (with length of report and data) to report back to HOST
Step 3	HOST reads the first 2 Bytes of DATA register to identify length of report and then reads remainder of data (length identified above) from DATA Register	

7.2.2.4 NOTES

- This is a mandatory request (if the DEVICE contains Input or Feature reports) and shall be supported on both the HOST end and the DEVICE end.
- This request is primarily used for getting Feature Report from the DEVICE.
- This request is also useful at initialization time for absolute input items. It is NOT intended for polling the data from the device and the HOST shall leverage the input register for this functionality.
- The DEVICE shall ignore a GET_REPORT requests with the REPORT TYPE set to Output, as it is not used in this specification.
- HOST will utilize repeated start for this transaction. DEVICE may use I^{2}C clock stretching per specification if critical.
- The protocol is optimized for Report <15. If a Report ID >= 15 is necessary, then the Report ID in the Low Byte must be set to '***1111***' and a Third Byte is appended to the protocol. This Third Byte contains the entire/actual report ID. The value ***1111*** in the Low Byte's Report ID is used as a sentinel value to indicate the need for the Report ID in the Third Byte. This mechanism optimizes the protocol for most reports (few devices need Report ID >=15) while still maintaining compatibility with other transports that allow Report IDs up to 255.

The following is an illustration of the protocol between the HOST and the DEVICE for GET_REPORT, in situations where Report ID needs to be < 15.

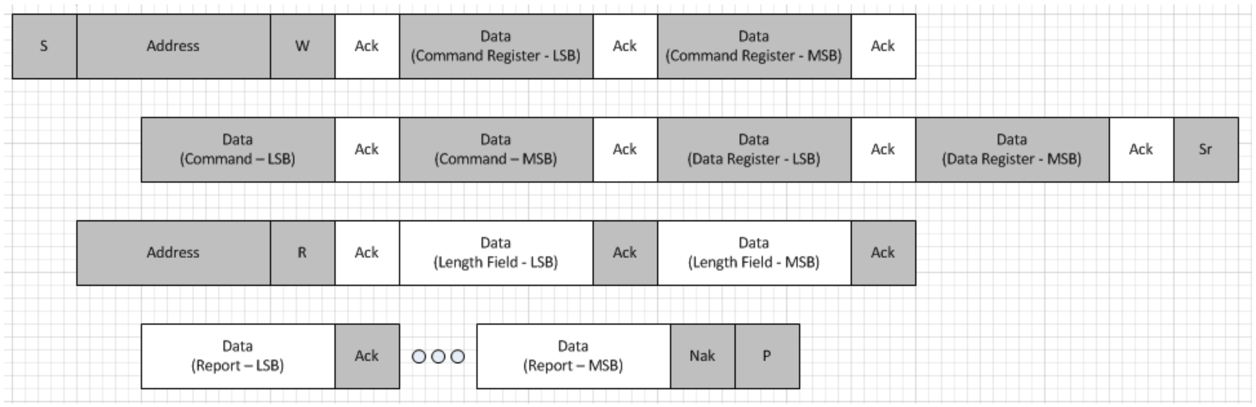
S
Address
W
Ack
Data (Command Register - LSB)
Ack
Data (Command Register - MSB)
Ack
Data (Command – LSB)
Ack
Data (Command – MSB)
Ack
Data (Data Register - LSB)
Ack
Data (Data Register - MSB)
Ack
Sr
Address
R
Ack
Data (Length Field - LSB)
Ack
Data (Length Field - MSB)
Ack
Data (Report – LSB)
Ack
Data (Report – MSB)
Nak
P

Figure 8: GET_REPORT Protocol Diagram (* Report ID < 15)

The following is an illustration of the protocol between the HOST and the DEVICE for GET_REPORT, in situations where Report ID needs to be >= 15.

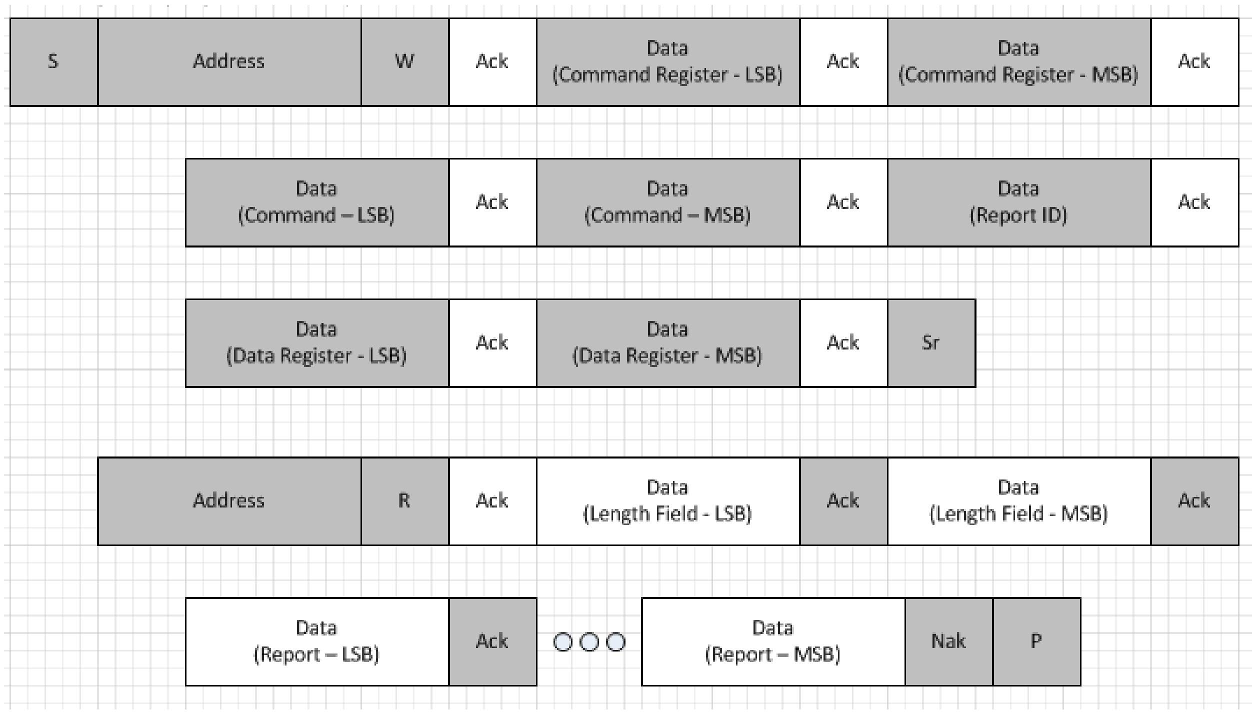
S
Address
W
Ack
Data (Command Register - LSB)
Ack
Data (Command Register - MSB)
Ack
Data (Command – LSB)
Ack
Data (Command – MSB)
Ack
Data (Report ID)
Ack
Data (Data Register - LSB)
Ack
Data (Data Register - MSB)
Ack
Sr
Address
R
Ack
Data (Length Field - LSB)
Ack
Data (Length Field - MSB)
Ack
Data (Report – LSB)
Ack
Data (Report – MSB)
Nak
P

Figure 9: GET_REPORT Protocol Diagram (*Report ID >= 15)

7.2.3 SET_REPORT

The SET_REPORT command is a specific request that the HOST may issue to the DEVICE at any time after initialization to set a singular report on the DEVICE. The DEVICE is responsible for accepting the value provided in the Data register and updating its state. A DEVICE must accept SET_REPORTS for all report types specified in the report descriptor.

7.2.3.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	0011b	This value is reserved for the SET_REPORT Command
Low Byte		
RESERVED	00b	This value is reserved and must be set to 00b
Report Type	{Output (10) | Feature (11)}	The HOST shall always set this to the specified value based on the report type the HOST is trying to set. The DEVICE shall honor and accept this report if the report type is supported in the Report Descriptor for the specified TLC.
Report ID	xxxxb	The HOST shall always set this to the specified value based on the TLC specific report the HOST is trying to set. The DEVICE shall honor this value per the rules outlined in the notes section below
Third Byte (Optional)		
Report ID	xxxxxxxxb	If the Report ID in the Low Byte is set to the sentinel value, then this optional Third Byte represents the Report ID. Please refer to Notes for details.

*all values ending with **'b'** indicate 'bits'

The HOST shall issue this to the command register and then fill the data register with the report being sent to the DEVICE. The Data shall be packaged as follows

- Length of Report (2 Bytes)
- Report including Report ID if defined in Report Descriptor (length identified above)

A HOST shall not send more data than is specified in the length of the report. For example, if the length of a sensor output report is 2 bytes, then the Data register should contain the length as 0x0004 followed by 2 bytes of the sensor device's output report.

7.2.3.2 RESPONSE

The DEVICE shall not need to respond back after receiving the data on the device register.

7.2.3.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST sends SET_REPORT to Command Register	
Step 2	HOST fills the Data Register with length and report and writes to the DEVICE.	
Step 3		DEVICE interprets the command and performs the necessary actions. The DEVICE does not need to respond back to HOST.

7.2.3.4 NOTES

- This is a mandatory request and shall be supported on both the HOST end and the DEVICE end.
- This request is useful for setting feature reports on a HID DEVICE.
- The request is also useful for setting output report on a HID DEVICE.
- The DEVICE might choose to ignore input SET_REPORT requests as meaningless.
- The protocol is optimized for Report <15. If a Report ID >= 15 is necessary, then the Report ID in the Low Byte must be set to '***1111***' and a Third Byte is appended to the protocol. This Third Byte contains the entire/actual report ID. The value ***1111*** in the Low Byte's Report ID is used as a sentinel value to indicate the need for the Report ID in the Third Byte. This mechanism optimizes the protocol for most reports (few devices need Report ID >=15) while still maintaining compatibility with other transports that allow Report IDs up to 255.

The following is an illustration of the protocol between the HOST and the DEVICE for SET_REPORT, in situations where Report ID needs to be < 15.

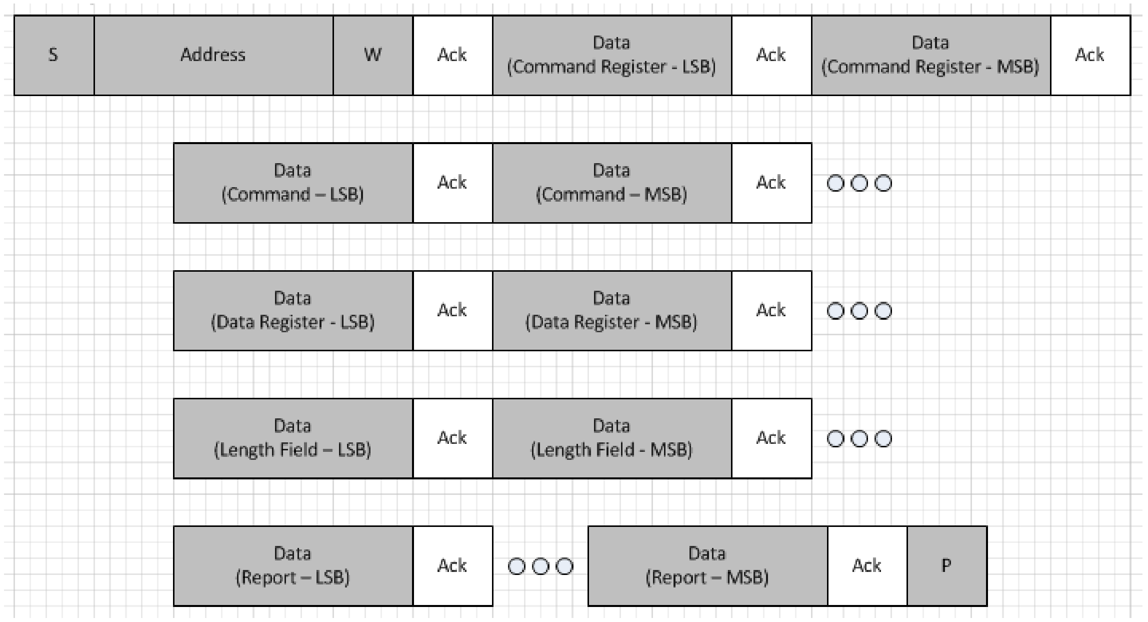
S
Address
W
Ack
Data (Command Register - LSB)
Ack
Data (Command Register - MSB)
Ack
Data (Command – LSB)
Ack
Data (Command – MSB)
Ack
Data (Data Register - LSB)
Ack
Data (Data Register - MSB)
Ack
Data (Length Field – LSB)
Ack
Data (Length Field - MSB)
Ack
Data (Report – LSB)
Ack
Data (Report – MSB)
Ack
P

Figure 10: SET_REPORT Protocol Diagram (*Report ID < 15)

The following is an illustration of the protocol between the HOST and the DEVICE for SET_REPORT, in situations where Report ID needs to be < 15.

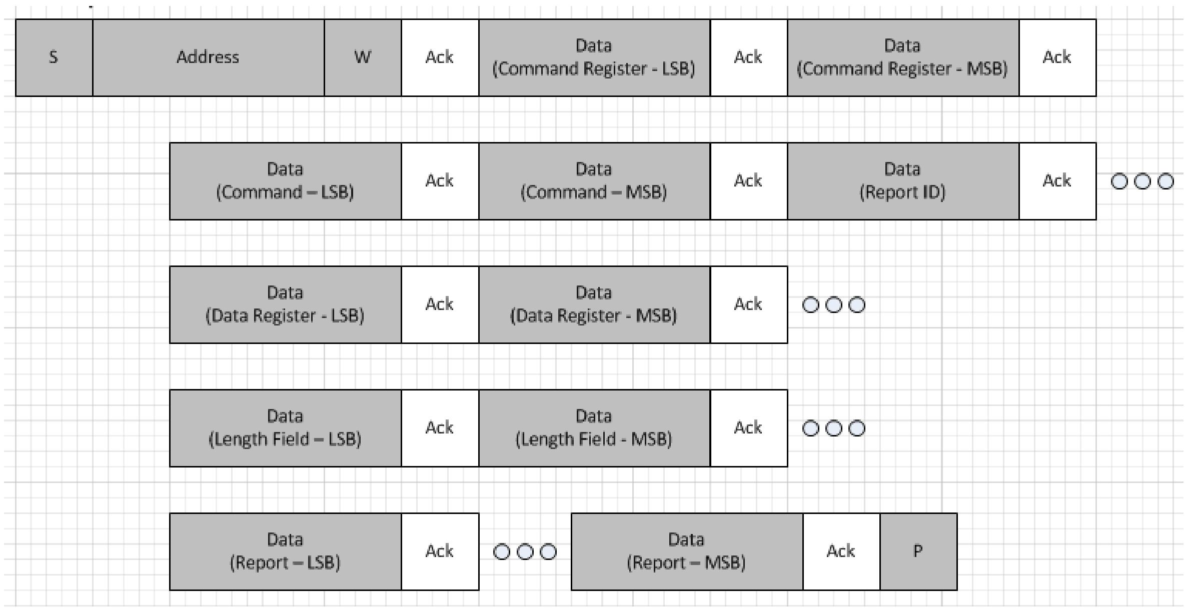
S
Address
W
Ack
Data (Command Register - LSB)
Ack
Data (Command Register - MSB)
Ack
Data (Command – LSB)
Ack
Data (Command – MSB)
Ack
Data (Report ID)
Ack
Data (Data Register - LSB)
Ack
Data (Data Register - MSB)
Ack
Data (Length Field – LSB)
Ack
Data (Length Field - MSB)
Ack
Data (Report – LSB)
Ack
Data (Report – MSB)
Ack
P

Figure 11: SET_REPORT Protocol Diagram (*Report ID >= 15)

7.2.4 GET_IDLE

The GET_IDLE command is a specific request that the HOST may issue to the DEVICE at any time after initialization to read the current idle rate for a particular input report. For more details see the SET_IDLE section below.

7.2.4.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	0100b	This value is reserved for the GET_IDLE Command
Low Byte		
RESERVED	00b	This value is reserved and must be set to 00b
Report Type	00b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.
Report ID	xxxxb	The HOST shall always set this to the TLC specific report whose Idle rate is being read.

*all values ending with **'b'** indicate 'bits'

7.2.4.2 RESPONSE

After the HOST sends the GET_IDLE command to the Command Register, the DEVICE must fill the DATA register with 2 Bytes of data to represent the reporting frequency and may stretch the clock for the subsequent HOST read per repeated start.

The Data shall be packaged as follows:

- Data Length (2 Bytes): Value = 4 Bytes
- Reporting Frequency Value (2 Bytes)

The report frequency will be interpreted as follows

- 0 = the duration is infinite
- 1..65535 = duration in mSec between reports.

7.2.4.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST sends GET_IDLE to Command Register	
Step 2		DEVICE fills the Data Register (with 2 Bytes of data representing the report frequency) to send to HOST
Step 3	HOST reads the first 2 Bytes of DATA register to identify length of report	
Step 4	HOST reads remainder of data (length identified above) from DATA Register	

7.2.4.4 NOTES

- This is an optional request and may be supported on either the HOST end or the DEVICE end. If both parties do NOT support this request, the request will be ignored.
- If the HOST implements the request but the DEVICE doesn't acknowledge within 5 seconds, then the HOST should interpret this as an unsupported feature on the device.
- It is the HOST's responsibility to leverage GET_IDLE and SET_IDLE if the HOST does not want the device to issue reports on a more frequent basis as long as the reports are not changing. GET and SET_IDLE were primarily used on keyboards with legacy HOSTs to control key repeat rates.
- HOST will utilize repeated start for this transaction. DEVICE may use I^{2}C clock stretching per specification.
- The protocol is optimized for Report <15. If a Report ID >= 15 is necessary, then the Report ID in the Low Byte must be set to '***1111***' and a Third Byte is appended to the protocol. This Third Byte contains the entire/actual report ID. The value ***1111*** in the Low Byte's Report ID is used as a sentinel value to indicate the need for the Report ID in the Third Byte. This mechanism optimizes the protocol for most reports (few devices need Report ID >=15) while still maintaining compatibility with other transports that allow Report IDs up to 255.

7.2.5 SET_IDLE

The SET_IDLE command is a specific request that the HOST may issue to the DEVICE at any time after initialization to set the report frequency on the DEVICE.

7.2.5.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	0101b	This value is reserved for the SET_IDLE Command
Low Byte		
RESERVED	00b	This value is reserved and must be set to 00b
Report Type	00b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.
Report ID	xxxxb	The HOST shall always set this to the TLC specific report whose Idle rate is being set.

*all values ending with **'b'** indicate 'bits'

The HOST must issue this to the command register after filling the data register with the report frequency being sent to the DEVICE.

A HOST shall not send more than 2 Bytes of data to the Data Register.

The Data shall be packaged as follows:

- Data Length (2 Bytes): Value = 4 Bytes
- Reporting Frequency Value in mSec (2 Bytes)

The report frequency that should be used must follow the value definitions below:

- 0 = the duration is infinite
- 1..65535 = duration in mSec between reports.

7.2.5.2 RESPONSE

The DEVICE shall not need to respond back after receiving the data on the device register.

7.2.5.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST fills the Data Register 2 Bytes of data to send to DEVICE.	
Step 2	HOST sends SET_IDLE to Command Register	
Step 3		DEVICE interprets the data in the DATA Register after receiving the Request in the COMMAND Register. The DEVICE does not need to respond back to HOST.

7.2.5.4 NOTES

- This is an optional request and may be supported on either the HOST end or the DEVICE end. If both parties do NOT support this request, the request will be ignored.
- It is the HOSTS responsibility to leverage GET_IDLE and SET_IDLE if the HOST does not want the device to issue reports on a more frequent basis.
- If the report ID is non-zero then the idle rate applies to the specified TLC.
- The protocol is optimized for Report <15. If a Report ID >= 15 is necessary, then the Report ID in the Low Byte must be set to '***1111***' and a Third Byte is appended to the protocol. This Third Byte contains the entire/actual report ID. The value ***1111*** in the Low Byte's Report ID is used as a sentinel value to indicate the need for the Report ID in the Third Byte. This mechanism optimizes the protocol for most reports (few devices need Report ID >=15) while still maintaining compatibility with other transports that allow Report IDs up to 255.

7.2.6 GET_PROTOCOL

The GET_PROTOCOL command is a specific request that the HOST may issue to the DEVICE at any time after initialization to identify the protocol mode the device is operating in. The DEVICE is responsible for responding with the current protocol it is operating in. All HIDs must initiate into report protocol and report this value if queried on initialization

7.2.6.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	0110b	This value is reserved for the GET_PROTOCOL Command
Low Byte		
RESERVED	00b	This value is reserved and must be set to 00b
Report Type	00b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.
Report ID	0000b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.

*all values ending with **'b'** indicate 'bits'

7.2.6.2 RESPONSE

After the HOST sends the GET_PROTOCOL command to the Command Register, the DEVICE must fill the value of the protocol in to the DATA Register and may stretch the clock for the subsequent HOST read per repeated start. The Data shall be packaged as follows

- Data Length (2 Bytes): Value = 4 Bytes
- Protocol Value (2 Byte)

The following are the defined protocol values:

- 0 = Boot Protocol
- 1 = Report Protocol
- 2-65535 = RESERVED

7.2.6.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST sends GET_PROTOCOL to Command Register	
Step 2		DEVICE fills the Data Register (with 2Bytes of data representing the protocol) to send to HOST
Step 3	HOST reads the first 2 Bytes of DATA register to identify length of report	
Step 4	HOST reads remainder of data (length identified above) from DATA Register	

7.2.6.4 NOTES

- This is an optional request and may be supported on either the HOST end or the DEVICE end. If both parties do NOT support this request, the request will be ignored.
- A DEVICE shall always default to Report Protocol on initialization unless set otherwise by HOST.
- This request is for compatibility with OLDER legacy HIDs and is rarely used on modern HOSTS.
- HOST will utilize repeated start for this transaction. DEVICE may use I^{2}C clock stretching per specification and per additional restrictions.

7.2.7 SET_PROTOCOL

The SET_PROTOCOL command is a specific request that the HOST may issue to the DEVICE at any time after initialization to set or change the protocol mode the device is operating in.

7.2.7.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	0111b	This value is reserved for the SET_PROTOCOL Command
Low Byte		
RESERVED	00b	This value is reserved and must be set to 00b
Report Type	00b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.
Report ID	0000b	The HOST shall always set this to the specified value. The DEVICE shall ignore this value and treat it as null.

*all values ending with **'b'** indicate 'bits'

The HOST must issue this to the command register after filling the data register with the protocol value being sent to the DEVICE.

A HOST shall not send more than 2 Bytes of data to the Data Register.

The Data shall be packaged as follows:

- Data Length (2 Bytes): Value = 4 Bytes
- Protocol Value (2 Bytes)

7.2.7.2 RESPONSE

The DEVICE shall not need to respond back after receiving the data on the device register.

7.2.7.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST fills the Data Register 2 Bytes of data to send to DEVICE.	
Step 2	HOST sends SET_PROTOCOL to Command Register	
Step 3		DEVICE interprets the data in the DATA Register after receiving the Request in the COMMAND Register. The DEVICE does not need to respond back to HOST.

7.2.7.4 NOTES

- This is an optional request and may be supported on either the HOST end or the DEVICE end. If both parties do NOT support this request, the request will be ignored.
- This request is for compatibility with OLDER legacy HIDs and is rarely used on modern HOSTS.

7.2.8 SET_POWER

The SET_POWER command is a specific request that the HOST may issue to the DEVICE at any time after initialization, to identify the Power State that the DEVICE should Transition to. The DEVICE must ensure that it transitions to the HOST specified Power State in under 1 second. All HID DEVICES must support this command though it is optional for HOSTS to support this command.

7.2.8.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	1000b	This value is reserved for the SET_POWER Command
Low Byte		
RESERVED	000000b	This value is reserved and must be set to 000000b
Power State	{On (00) | SLEEP (01)}	The HOST shall always set this to the Power State that the DEVICE is required to transition to.

*all values ending with **'b'** indicate 'bits'

Please note that the Low Byte uses a special format in the case of SET_POWER Request. The Low byte contains the Power State.

The following are the defined Power State values:

- 00 = ON
- 01 = SLEEP
- 1x = RESERVED

There is no value sent to the Data Register.

7.2.8.2 RESPONSE

The DEVICE shall not respond back after receiving the command. The DEVICE is mandated to enter that power state imminently.

If the DEVICE wishes to wake the HOST from its low power state, it can issue a wake by asserting the interrupt.

7.2.8.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST sends SET_POWER to Command Register	
Step 2		DEVICE interprets the command and transitions to the appropriate power state. The DEVICE does not need to respond back to HOST.

7.2.8.4 NOTES

- This is an optional request for HOSTs but mandatory for DEVICEs.
- Once the host issues a SET_POWER(ON) command, the DEVICE must transition to the ON state immediately. Clock stretching can be employed (up to the maximum defined limit in the "Sizes and Constants" section) if needed.

The following is an illustration of the protocol between the HOST and the DEVICE for SET_POWER.

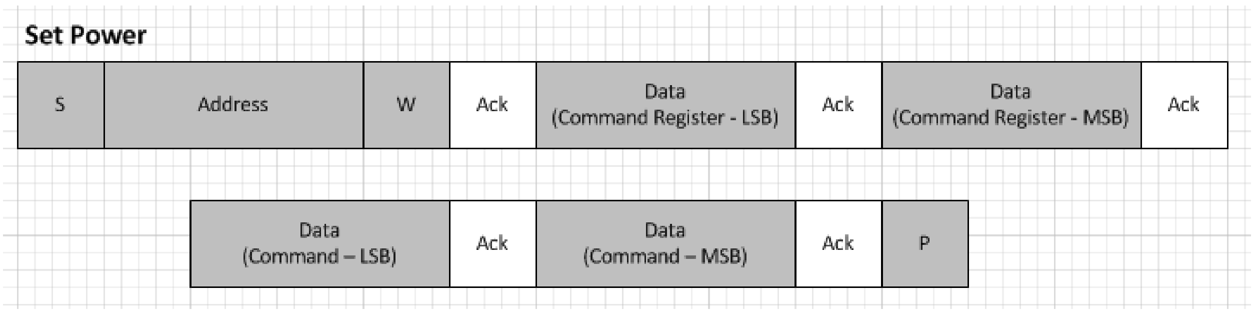

Figure 12: SET_POWER Protocol Diagram

7.2.9 RESERVED COMMAND RANGE

The HOST shall not issue a command to the device from the reserved range. Should a device receive such a command, it should ignore it.

7.2.9.1 REQUEST

Data	Value	Remarks
High Byte		
RESERVED	0000b	This value is reserved and must be set to 0000b
OpCode	Reserved Ranges	This value is reserved and shall not be used by HOST. Please refer to Section 7.2 for the reserved ranges.
Low Byte		
Report Type	N/A	If the OpCode is from a reserved range, the value for Report Type shall be ignored.
Report ID	N/A	If the OpCode is from a reserved range, the value for Report ID shall be ignored.

*all values ending with **'b'** indicate 'bits'

7.2.9.2 RESPONSE

If the HOST sends a request with the opcode in the reserved range, the device shall ignore the request from the HOST and treat it as invalid. The device shall not respond back to the HOST.

7.2.9.3 SEQUENCE OF OPERATIONS

The following is the sequence of operations on the HOST end and the DEVICE end for this specific request:

Sequence	HOST Side	DEVICE Side
Step 1	HOST sends RESERVED request type to Command Register	
Step 2		DEVICE will ignore this request and not respond back to HOST with any data

7.3 INTERRUPT SERVICING

The following diagram identifies the procedure the HOST must follow when servicing an interrupt.

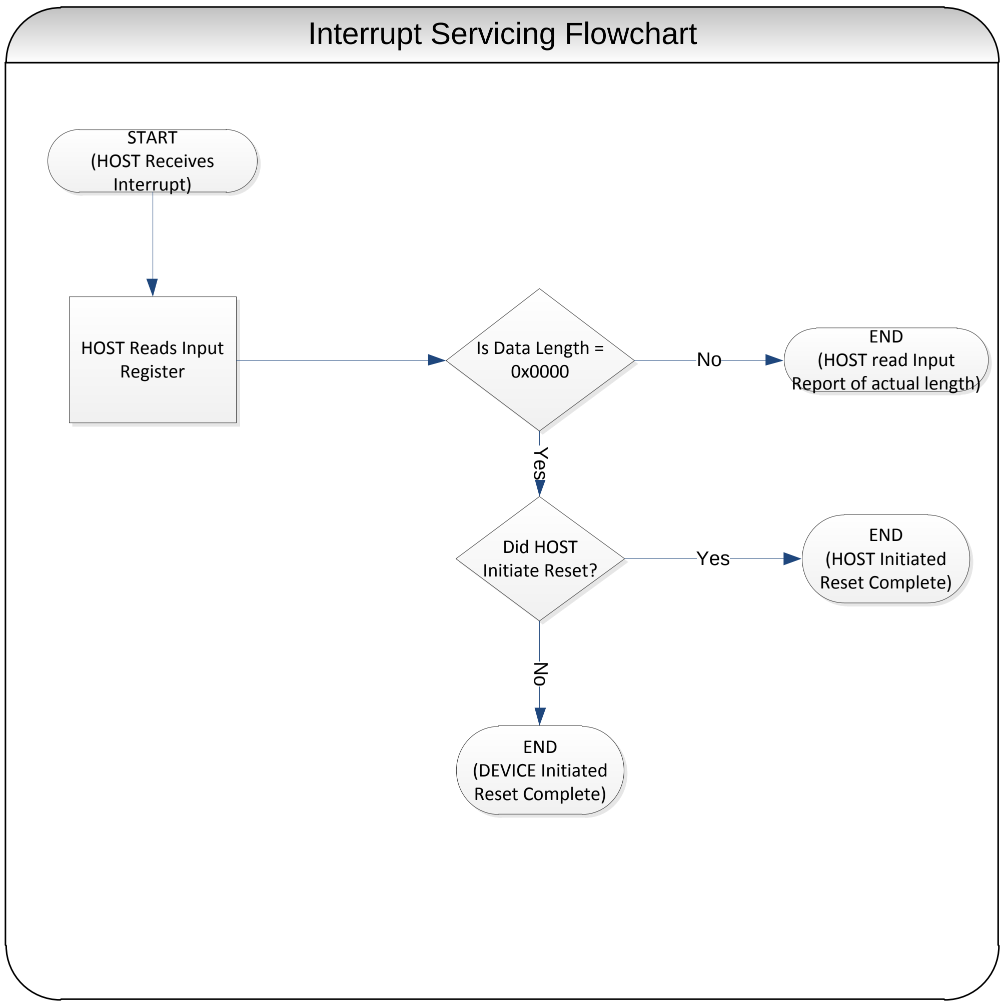
Interrupt Servicing Flowchart
START (HOST Receives Interrupt)
HOST Reads Input Register
Is Data Length = 0x0000
No
END (HOST read Input Report of actual length)
Yes
Did HOST Initiate Reset?
Yes
END (HOST Initiated Reset Complete)
No
END (DEVICE Initiated Reset Complete)

Figure 13: Interrupt Servicing Flowchart

7.4 INTERRUPT LINE MANAGEMENT

DEVICE is responsible to assert the interrupt line (level trigger interrupt required) when it has data.

DEVICE must keep the interrupt line asserted until the data has been fully read by the HOST. After this point, the DEVICE must de-assert the interrupt line if it has no more data to report to the HOST.

When the DEVICE has new data to report, it must repeat the process above.

8 POWER MANAGEMENT

The following section identifies the details around HOST and DEVICE power management. The following sections identify the two specific scenarios for power management:

- Device Initiated Power Optimizations
- Host Initiated Power Optimizations

8.1 DEVICE INITIATED POWER OPTIMIZATIONS (DIPO)

The DEVICE is responsible for optimizing its power utilization in the absence of any power settings from the HOST. This will enable the DEVICE to enter its lowest power state on its own (i.e. without HOST intervention) while ensuring that the DEVICE is able to get and set all requests with the HOST in a timely manner.

To correctly comply with Device Initiated Power Optimizations, the following shall be observed:

- The DEVICE is responsible for preserving its state across its low power mode(s).
- All DEVICE power optimizations must be transparent to the HOST and end user(s).
- The DEVICE shall respond to all requests from HOST in a timely manner (i.e. the low power mode(s) shall be transparent to the HOST). The DEVICE is responsible for bringing itself to higher power modes on user or system interactions in a timely manner.
- The DEVICE shall notify the HOST on any INPUT report changes in a lossless manner (i.e. no events should be lost, or deleted by the DEVICE).
- The power states described in HIPO do NOT apply to DIPO.

Scenarios where DEVICE Initiated Power Optimizations are generally deployed include the following scenarios

- ***Scenario 1 – DEVICE is idle for a short interval of time***: The DEVICE determines that it is idle (e.g. accelerometer is lying flat on a table and there is no measurable change in acceleration) and puts itself in to its lowest power state where it reduces its internal sensing frequency until motion is reinitiated. As soon as motion starts, data is immediately sent to host.
- ***Scenario 2 – DEVICE reduces its sensing frequency***: The DEVICE reduces the frequency at which it scans for data (e.g. if the digitizer does not sense a human finger is present, it samples at 10Hz as compared to 100Hz. However once user interaction is detected, it increases its sensing interval).

8.2 HOST INITIATED POWER OPTIMIZATIONS (HIPO)

The HOST is responsible for optimizing the power of the overall system and the DEVICE. This method of power optimization is to be used when the HOST wishes to provide power optimization notifications to devices.

The following power states are defined for HIPO and are not to be confused with vendor specific DIPO states.

- ON
- SLEEP

There is no need for an OFF state definition. The DEVICE is responsible for being in the ON state when HID I^{2}C communications are initiated.

The table below identifies the properties a DEVICE and a HOST must follow

Power State	HOST Responsibility	DEVICE Responsibility
ON	• The HOST is responsible for addressing interrupts and issues commands to the device as necessary.	• The DEVICE is responsible for being in the ON power state after Reset / Initialization. • The DEVICE must respond to a SET POWER command from the HOST.
SLEEP	• The HOST is responsible for informing the device to enter SLEEP state. • The HOST is responsible for setting the device into ON state if the DEVICE alerts via the interrupt line. • If a HOST needs to communicate with the DEVICE it MUST issue a SET POWER command (to ON) before any other command.	• The DEVICE must de-assert the interrupt line if asserted, before HIPO. • The DEVICE may send an interrupt to the HOST to request servicing (e.g. DEVICE wishes to remote wake the HOST) • The DEVICE must reduce the power draw to an absolute minimum to maintain state and optionally support remote wake. • The DEVICE must respond to a SET POWER command from the HOST.

Scenarios where HOST Initiated Power Optimizations (HIPO) are generally deployed include the following scenarios

- ***Scenario 1 – No application communication with the device (Idle Detection)***: The HOST determines that there are no active applications that are currently using the specific HID DEVICE. The HOST is recommended to issue a HIPO command to the DEVICE to force the DEVICE in to a lower power state (e.g. there is no application needed to talk to the gyroscope, so the HOST informs the gyroscope to go into SLEEP state).
- ***Scenario 2 – System Suspend***: The HOST is going into a deep power optimized state and wishes to put all the devices into a low power state also. The HOST is recommended to issue a HIPO

command to the DEVICE to force the DEVICE in to a lower power state (e.g. a PC enters standby state and wishes to put the consumer control buttons on the keyboard to SLEEP power state).

The following figure shows the allowed DEVICE power state transitions.

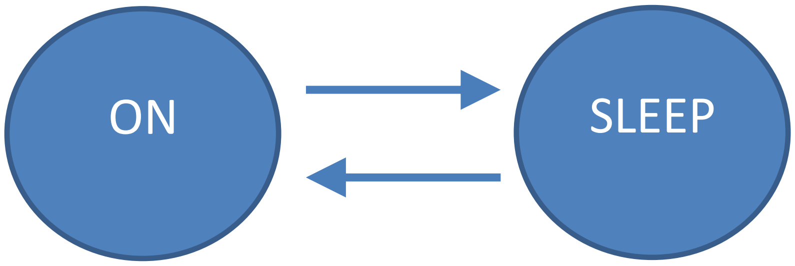

Figure 14: DEVICE Power State Transitions

Notes:

- It is the HOST responsibility to ensure that the HOST Initiated Power Optimization commands comply with the allowed state transitions.
- On the completion of the RESET operation, the DEVICE should reside in the ON Power State.
- The DEVICE is recommended to transition to the new Power State in a reasonable amount of time. This may vary between device types but should never exceed 1sec. Having this limit ensures that HOSTS can power optimize their power in a reasonable timeframe.
- If the DEVICE receives a Power Setting that it is already in (e.g. DEVICE is in ON Power State, and receives a SET_POWER command to turn to ON), the device need not take any action.
- The DEVICE only goes into the OFF power state when the HOST is completely shut down.

9 ERROR HANDLING

The following section summarizes the error detection and handling procedures that the HOST and the DEVICE must follow. Errors on the I^2C bus for the purposes of this specification will be broken out in to the following categories:

- Protocol Errors
- Timeout Errors

9.1 PROTOCOL ERRORS

Protocol errors are possible on I^2C and are more frequent on a poorly designed system. To guard against basic protocol errors, they are further characterized into the following classifications for the purposes of this specification:

- Short Packet Errors
- Bit Level Errors

9.1.1 SHORT PACKET ERRORS

Short packet errors occur when the HOST or the DEVICE does not return the number of bits as identified in the I^2C protocol request and length field. The software driver stack is responsible for guarding against this situation. Short packet errors must be handled by the HOST and preferably discarded by the HOST driver stack.

9.1.2 BIT LEVEL ERRORS

Bit level errors may occur on the I^2C bus. These errors are generally as a result of noise on the bus or interference from other buses in the system. This specification does not support CRC or other detection mechanism for single/multiple bit errors on the I^2C data line.

However, it is often possible for the HOST parser to identify a malformed report and discard it. It is the responsibility of the HOST HID driver stack to guard against a malformed report that is not conformant with the Report Descriptor.

9.2 TIMEOUT ERRORS

The HID over I^{2}C protocol is sequential with the expectation that all device transmissions need to be completed in the timely fashion. In most cases, the responses from DEVICE to HOST will complete in a matter of milliseconds. In the event that the DEVICE is stuck and is unable to revert itself, there is a forced timeout delay after which the HOST is required to RESET the device and restart operations. A HOST should not need to execute on this command unless there is an error in the state machine of the device or the I^{2}C bus itself is being held up by another device (a rare condition).

TIMEOUT_HostInitiatedReset = 5 seconds.

Different HOSTs may allow for proprietary methods to adjust the value of this timeout for their specific devices but it is mandatory for the HOST to support a timeout value (if optimized for the specific system).

The HID over I^{2}C spec does allow the DEVICE to leverage clock stretching at the I^{2}C protocol level, if the DEVICE is not ready to respond. Clock stretching is a common practice for many I^{2}C DEVICEs but it does significantly decrease the performance of the I^{2}C bus as a whole (especially when there are multiple I^{2}C DEVICES attached to the same controller. To reduce the impact of clock stretching, the HID I^{2}C spec requires an upper bound to clock stretching.

TIMEOUT_ClockStrectch = 10 milliseconds.

DEVICEs are requested to limit the occasions when they deploy clock stretching to key scenarios around power management and command response only.

10 ACPI SPECIFIC DETAILS

10.1 ACPI EXTENSIONS AROUND HID

The ACPI 5.0 Spec (ACPI) will include support for HID Class Devices. Specifically, the ACPI definitions for HID I2C are defined in the following table:

Field	Value	M / O	ACPI object	Format	Comments
Hardware ID	Vendor Specific	M	_HID	String in the format of VVVVdddd (e.g. MSFT0011)	VendorID + DeviceID
Compatible ID	[PNP0C50]	M	_CID	String in the format of ACPIxxxx or PNPxxxx	CompatibleID
Subsystem	Vendor Specific	O	_SUB	String in the format of VVVVssss (e.g MSFQ1234)	SubVendorID + SubSystemID
Hardware Revision	Vendor Specific	M	_HRV	0xRRRR (2byte revision)	RevisionID
Current Resource Settings	Vendor Specific	M	_CRS	Byte stream	-I2CSerialBusConnection for access to the device -GpioInt for interrupts.
Device Specific Method	[3CDFF6F7-4267-4555-AD05-B30A3D8938DE]	M	_DSM	Package	This GUID defines a structure that contains the HID Descriptor address (2 Bytes).

For additional guidance in related specifications, please refer to the ACPI V5.0 specification and the "*Windows ACPI Design Guide for SOC Platforms*".

11 SIZES AND CONSTANTS

The following section identifies the minimum & maximum sizes of various parameters:

Data Element	Minimum Value	Maximum Value	Notes
Report Descriptor	0 Bytes	65535 Bytes (2^{16}-1)	The maximum length is determined by the size of ***wReportDescLength*** in the HID Descriptor.
Report (Input, Output)	0 Bytes	65535 Bytes (2^{16}-1)	The maximum length is determined by the size of ***wReportDescLength*** in the HID Descriptor.
Report IDs	0	255	The current HID I^{2}C Specification is optimized for Report IDs<15. Report IDs >15 are supported via the Third Byte.
Max Clock Stretching	0 mSec	10 mSec	The maximum length in time that a DEVICE should clock stretch between Write-Read I^{2}C operations.

*While the I^{2}C spec does curtail the maximum length of an input report to 65535 Bytes, many other transports support short packets. It is recommended to have Input and Output Reports shorter than 63 Bytes to work efficiently on some Hosts, as this was a restriction in the original HID specification.

The following table identifies the average size of a report descriptor and the pertinent reports (input, output, and feature) for that device type. This information is NOT specific to I^{2}C and should be roughly identical on any HID Transport.

Device Type	Report Descriptor	Input Report	Output Report	Feature Descriptor
Basic Keyboard	50-60 Bytes	~8 Bytes	~1 Byte	0 Byte
Keyboard with consumer controls)	~200 Bytes	~8 + 2 Bytes*	1 Byte	0 Byte
Basic Mouse	~40 Bytes	~5 Bytes	0 Byte	0 Byte
Accelerometer Sensor	~80 Bytes	~6 Bytes	0 Byte	~4 Bytes
Touchscreen	~100 – 500 Bytes	~20 – 40** Bytes	0 Byte	~2 Bytes

*These devices generally use multiple Top-Level Collections
**This is based on 2-finger touch points per report.

12 SPEC UPDATE SUMMARY

This section summarizes the main updates made to the protocol specification as it has advances from V0.9 forward. The objective of this section is to help a reader identify which sections of the specification have undergone major changes between revisions.

12.1 UPDATES BETWEEN V0.9 -> V0.91:

The following section summarizes the updates made between V0.9 and V0.91:

- New Sections/Features added
  - Support for Report IDs >= 15 in Get/Set_Report.
- Changes to Existing Sections:
  - Fixed errors in section 13.4 – 13.6. Added a note to indicate that there are ACK bits in diagrams for section 13.3 & 13.5.
  - Updated "*wMaxInputLength*" min value from 0 to 2.
  - Update the Get/Set_Report features to implement Write-Read instead of Write-Write-Read.
  - Traces removed from the Appendix at the end and integrated into the right section of spec.

12.2 UPDATES BETWEEN V0.91 -> V1.0:

The following section summarizes the updates made between V0.91 and V1.0:

- New Sections/Features added
  - N/A
- Changes to Existing Sections:
  - Verbiage around maximum clock stretching recommendation (section 11)
  - VendorID field in HID Descriptor must be non-zero.
  - Level triggered interrupts changed to "required" from "recommended".
  - Addressed error in section 4.3 to clarify Byte ordering (not bit ordering).
  - Removed errors in section 7.2.8.3 referring to Set_Protocol.
  - Updated errors in appendix section 13.3 and 13.7
  - Updated the license agreement to point to Microsoft Open Specifications – Community Promise.
  - Removed Windows specific FAQs from specification.
  - Removed confidential watermark.

13 EXAMPLE: SAMPLE ACCELEROMETER SENSOR

This example demonstrates how to build a HID Compliance Accelerometer Sensor over I^{2}C.

13.1 ASL LAYOUT

The following table identifies the ASL layout for the Accelerometer Device.

```
Scope (\_SB) {

//--------------------
//  General Purpose I/O, ports 0...127
//--------------------


Device(HIDI2C_DEVICE1) {
    Name(_ADR,0)
    Name (_HID, "MSFT1234”)
    Name (_CID, "PNP0C50")
    Name (_UID, 3)

    Method(_CRS, 0x0, NotSerialized)
    {
        Name (RBUF, ResourceTemplate ()
        {
        // Address 0x07 on I2C-X (OEM selects this address)
        //IHV SPECIFIC I2C3 = I2C Controller; TGD0 = GPIO Controller;
        I2CSerialBus (0x07, ControllerInitiated, 100000,AddressingMode7Bit, "\\_SB.I2C3",,,,)
        GpioInt(Level, ActiveLow, Exclusive, PullUp, 0, "\\_SB. TGD0", 0 , ResourceConsumer, , )
{40}
        })
     Return(RBUF)
     }



     Method(_DSM, 0x4, NotSerialized)
     {
        // BreakPoint
        Store ("Method _DSM begin", Debug)

        // DSM UUID
        switch(ToBuffer(Arg0))
        {
            // ACPI DSM UUID for HIDI2C
            case(ToUUID("3CDFF6F7-4267-4555-AD05-B30A3D8938DE"))
            {
                // DSM Function
                switch(ToInteger(Arg2))
                {
                    // Function 0: Query function, return based on revision
                    case(0)
                    {
                        // DSM Revision
                        switch(ToInteger(Arg1))
                        {
                            // Revision 1: Function 1 supported
                            case(1)
                            {
                                Store ("Method _DSM Function Query", Debug)
                                Return(Buffer(One) { 0x03 })
                            }

                            default
                            {
                                // Revision 2+: no functions supported
```

```
                              Return(Buffer(One) { 0x00 })
                          }
                      }
                  }

                  // Function 1 : HID Function
                  case(1)
                  {
                      Store ("Method _DSM Function HID", Debug)

                      // HID Descriptor Address
                      Return(0x0001)
                  }

                  default
                  {
                      // Functions 2+: not supported
                  }
              }
          }

          default
          {
              // No other GUIDs supported
              Return(Buffer(One) { 0x00 })
          }
      }
  }
}
```

Notes:

- The *_CID* field represents the compatible ID for the device and must always be set to “PNP0C50” for HID I^{2}C compliant DEVICE. This enables the HOST to identify the DEVICE’s class and load the correct software (driver) without any additional software from the device manufacturer.
- *GpioInt* identifies the example of a GPIO based interrupt (and its characteristics) associated with the DEVICE.
- *The _DSM field represents a device specific method that is associated with the HID over I^{2}C specification. Additional Details on _DSM are available in the ACPI specification (Section 9.14.1). The HID _DSM GUID is* "3CDFF6F7-4267-4555-AD05-B30A3D8938DE". It is also required to have 1 function (HID) associated with this _DSM that returns a 2-Byte value for the HID Descriptor address. For additional details, please refer to the ACPI Design Guide for SoC Platforms.
- The HID Descriptor Address in this example DEVICE is **0x0001**.

13.2 HID DESCRIPTOR

The HID Descriptor for the Accelerometer Device is described below.

Byte Offset	Field	VALUE (HEX)	Notes
0	**wHIDDescLength**	001E	Length of HID Descriptor. Fixed to 30 Bytes (0x0018)
2	**bcdVersion**	0100	Compliant with Version 1.00
4	**wReportDescLength**	00E5	Report Descriptor is 229 Bytes (0x00E5)
6	**wReportDescRegister**	0002	Identifier to read Report Descriptor
8	**wInputRegister**	0003	Identifier to read Input Report
10	**wMaxInputLength**	000B	Input Report is 9 Bytes + 2 Bytes length field
12	**wOutputRegister**	0004	Identifier to read Output Report
14	**wMaxOutputLength**	0000	No Output Report
16	**wCommandRegister**	0005	Identifier for Command Register
18	**wDataRegister**	0006	Identifier for Data Register
20	**wVendorID**	049F	Vendor ID 0x049F
22	**wProductID**	0101	Device ID 0101
24	**wVersionID**	0100	Version 1.00
26	**RESERVED**	00 00 00 00	RESERVED

13.3 HID DESCRIPTOR RETRIEVAL

The following section identifies how the HID Descriptor is retrieved for this DEVICE.

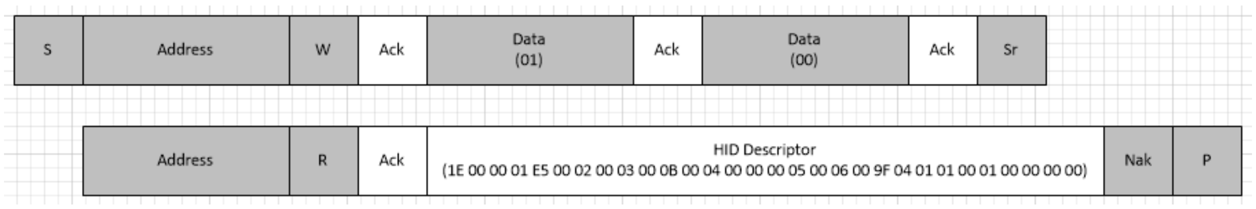
S
Address
W
Ack
Data (01)
Ack
Data (00)
Ack
Sr
Address
R
Ack
HID Descriptor (1E 00 00 01 E5 00 02 00 03 00 0B 00 04 00 00 00 05 00 06 00 9F 04 01 01 00 01 00 00 00 00)
Nak
P

Figure 15: HID Descriptor Retrieval Example

**To simplify the diagram, individual ACKs/nACKs are not shown within the Descriptor, though they will be used in the protocol as per the I^{2}C definitions.*

13.4 REPORT DESCRIPTOR

The Report Descriptor comprises of the new usages approved in HID RR #39, as part of the HID Sensors Usage Page.

```
0x05, 0x20,                 //Sensor Usage Page
0x09, 0x73,                 //HID_USAGE_SENSOR_TYPE_MOTION_ACCELEROMETER_3D,
0xa1, 0x00,                 //HID_COLLECTION(Physical),
//<FEATURE>
0x05, 0x20,                 //HID_USAGE_PAGE_SENSOR
0x0a, 0x16, 0x03,           //HID_USAGE_SENSOR_PROPERTY_REPORTING_STATE
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,           //LOGICAL_MAXIMUM (255)
0x75, 0x08,                 //HID_REPORT_SIZE(8)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x03, 0x03,           //HID_USAGE_SENSOR_PROPERTY_SENSOR_STATUS
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,           //LOGICAL_MAXIMUM (255)
0x75, 0x08,                 //HID_REPORT_SIZE(8)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x09, 0x03,           //HID_USAGE_SENSOR_PROPERTY_SENSOR_CONNECTION_TYPE
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,           //LOGICAL_MAXIMUM (255)
0x75, 0x08,                 //HID_REPORT_SIZE(8)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x0f, 0x03,           //HID_USAGE_SENSOR_PROPERTY_CHANGE_SENSITIVITY_ABS
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,//LOGICAL_MAXIMUM (65535)
0x75, 0x10,                 //HID_REPORT_SIZE(16)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0x65, 0x1a,                 //HID_USAGE_SENSOR_UNITS_G
0x55, 0x02,                 //HID_UNIT_EXPONENT(2)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x0e, 0x03,           //HID_USAGE_SENSOR_PROPERTY_REPORT_INTERVAL
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0xff, 0xff,   //LOGICAL_MAXIMUM (4294967295)
0x75, 0x20,                 //HID_REPORT_SIZE(32)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0x65, 0x19,                 //HID_USAGE_SENSOR_UNITS_MILLISECOND
0x55, 0x00,                 //HID_UNIT_EXPONENT(0)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x52, 0x24,
//HID_USAGE_SENSOR_DATA(HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION,HID_USAGE_SENSOR_DATA_MOD_MAX)
0x16, 0x01, 0x80,           //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,           //LOGICAL_MAXIMUM (32767
0x75, 0x10,                 //HID_REPORT_SIZE(16)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0x65, 0x1a,                 //HID_USAGE_SENSOR_UNITS_G
0x55, 0x02,                 //HID_UNIT_EXPONENT(2)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
0x0a, 0x52, 0x34,
//HID_USAGE_SENSOR_DATA(HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION,HID_USAGE_SENSOR_DATA_MOD_MIN)
0x16, 0x01, 0x80,           //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,           //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                 //HID_REPORT_SIZE(16)
0x95, 0x01,                 //HID_REPORT_COUNT(1)
0x65, 0x1a,                 //HID_USAGE_SENSOR_UNITS_G
0x55, 0x02,                 //HID_UNIT_EXPONENT(2)
0xb1, 0x02,                 //HID_FEATURE(DATA_VAR_ABS)
//</FEATURE>
//<INPUT>
0x05, 0x20,                 //HID_USAGE_PAGE_SENSOR
0x0a, 0x01, 0x02,           //HID_USAGE_SENSOR_STATE
0x15, 0x00,                 //LOGICAL_MINIMUM (0)
```

```
0x26, 0xff, 0x00,          //LOGICAL_MAXIMUM (255)
0x75, 0x08,                //HID_REPORT_SIZE(8)
0x95, 0x01,                //HID_REPORT_COUNT(1)
0x81, 0x02,                //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x02, 0x02,          //HID_USAGE_SENSOR_EVENT
0x15, 0x00,                //LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,          //LOGICAL_MAXIMUM (255)
0x75, 0x08,                //HID_REPORT_SIZE(8)
0x95, 0x01,                //HID_REPORT_COUNT(1)
0x81, 0x02,                //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x53, 0x04,          //HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION_X_AXIS
0x16, 0x01, 0x80,          //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,          //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                //HID_REPORT_SIZE(16)
0x95, 0x01,                //HID_REPORT_COUNT(1)
0x65, 0x1a,                //HID_USAGE_SENSOR_UNITS_G
0x55, 0x0e,                //HID_UNIT_EXPONENT(14)
0x81, 0x02,                //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x54, 0x04,          //HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION_Y_AXIS
0x16, 0x01, 0x80,          //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,          //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                //HID_REPORT_SIZE(16)
0x95, 0x01,                //HID_REPORT_COUNT(1)
0x65, 0x1a,                //HID_USAGE_SENSOR_UNITS_G
0x55, 0x0e,                //HID_UNIT_EXPONENT(14)
0x81, 0x02,                //HID_INPUT(DATA_VAR_ABS)
0x0a, 0x55, 0x04,          //HID_USAGE_SENSOR_DATA_MOTION_ACCELERATION_Z_AXIS
0x16, 0x01, 0x80,          //LOGICAL_MINIMUM (-32767)
0x26, 0xff, 0x7f,          //LOGICAL_MAXIMUM (32767)
0x75, 0x10,                //HID_REPORT_SIZE(16)
0x95, 0x01,                //HID_REPORT_COUNT(1)
0x65, 0x1a,                //HID_USAGE_SENSOR_UNITS_G
0x55, 0x0e,                //HID_UNIT_EXPONENT(14)
0x81, 0x02,                //HID_INPUT(DATA_VAR_ABS)
//Shake Event Notification
0x0a, 0x51, 0x04,          // USAGE (Data: Motion Intensity)
0x15, 0x00,                //LOGICAL_MINIMUM (0)
0x25, 0x40,                //LOGICAL_MAXIMUM (64)
0x75, 0x08,                //HID_REPORT_SIZE(8)
0x95, 0x01,                //HID_REPORT_COUNT(1)
0x81, 0x02,                //HID_INPUT(DATA_VAR_ABS)
//</INPUT>
0xc0                       //HID_END_COLLECTION
```

13.5 REPORT DESCRIPTOR RETRIEVAL

The following section identifies how the Report Descriptor is retrieved for this DEVICE.

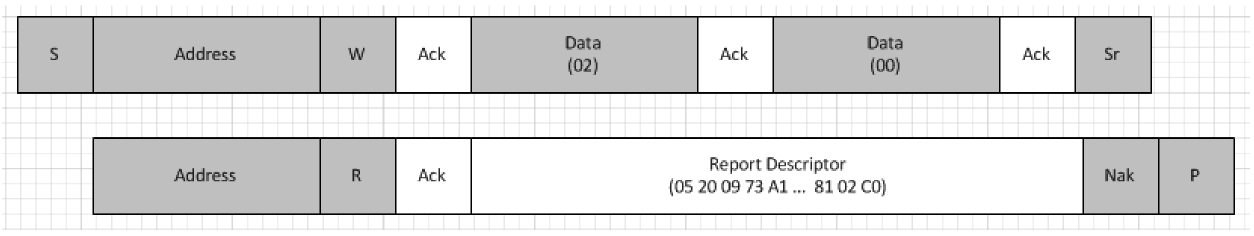

Figure 16: Report Descriptor Retrieval Example

**To simplify the diagram, individual ACKs/nACKs are not shown within the Report Descriptor, though they will be used in the protocol as per the I²C definitions.*

13.6 REPORTS

The Following table summarizes the report format for each of the three reports for this device.

Input Report [9 Bytes]	**Output Report [0 Bytes]**	**Feature Report [13 Bytes]**
Sensor State (1 Byte) Sensor Event (1Byte) Acceleration Axis X (2 Bytes) Acceleration Axis Y (2 Bytes) Acceleration Axis Z (2 Bytes) Data: Motion Intensity (1 Byte)	N/A	Property: Reporting State (1 Byte) Property: Sensor Reporting State (1 Byte) Property: Connection Type (1Byte) Property: Change Sensitivity Abs (2 Bytes) Property: Report Interval (4 Bytes) Property: Acc Mod Max (2 Byte) Property: Acc Mod Min (2 Byte)

13.7 READING AN INPUT REPORT

The following section identifies how the HOST should read an Input Report from the DEVICE.

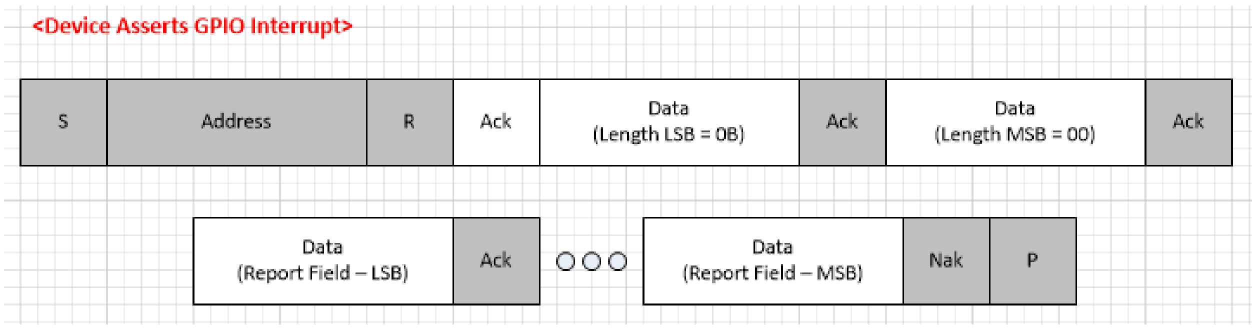
<Device Asserts GPIO Interrupt>
S
Address
R
Ack
Data (Length LSB = 0B)
Ack
Data (Length MSB = 00)
Ack
Data (Report Field – LSB)
Ack
Data (Report Field – MSB)
Nak
P

Figure 17: Input Report Retrieval Example