GDS® NOTE ACCEPTOR: COMMUNICATION PROTOCOL V1.3 DRAFT 6

Gaming Device Standards

Implemented as USB HID Class Device

Gaming Standards Association
GDS Technical Committee

Released: TBD

GAMINGSTANDARDS.COM

GDS® Note Acceptor: Communication Protocol v1.3 DRAFT 6

Released TBD, by Gaming Standards Association (GSA).

Patents and Intellectual Property

NOTE: The user's attention is called to the possibility that compliance with this [standard/specification] may require use of an invention covered by patent rights. By publication of this [standard/specification], GSA takes no position with respect to the validity of any such patent rights or their impact on this [standard/specification]. Similarly, GSA takes no position with respect to the terms or conditions under which such rights may be made available from the holder of any such rights. Contact GSA for further information.

Trademarks and Copyright

GSA Contact Information

E-mail: sec@gamingstandards.com

WWW: http://www.gamingstandards.com

Table of Contents

I About This Document

This specification defines the communication protocol of the GSA GDS (Gaming Device Standards) note acceptor.

I.I GDS (Gaming Device Standards)

GSA Gaming Device Standards controls the flow of information between an electronic gaming machine (EGM) and the array of peripheral devices operating inside it, including bill validators, card readers and ticket printers using Universal Serial Bus (USB) standards protocol. In essence, each peripheral device uses one command set to communicate with its host machine. That information can then be relayed to the casino management system through the machine message protocol, such as GSA's G2S™ (Game-to-System) Message Protocol.

For more details about GSA, visit the Web site: http://www.gamingstandards.com.

I.II Acknowledgements

The Gaming Standards Association would like to express its appreciation to all members of the GDS committee, past and present, for their significant contribution and dedication to the creation of this standard.

I.III Related Documents

I.III.I GSA

Gaming Standards Association documents referenced (http://www.gamingstandards.com):

Table I.1 Referenced GSA Documents

Ref	Title and Source
BCTS	GDS Bar Coded Ticket Format Specification

I.III.II USB

USB Implementers Forum, Inc., http://www.usb.org, documents referenced:

Table I.2 Referenced USB Documents

Ref	Title and Source
USB 2.0	Universal Serial Bus Specification, v2.0 http://www.usb.org/developers/docs/
DCDHID	Device Class Definition for HID, v1.11 http://www.usb.org/developers/hidpage/
DFU	Device Firmware Upgrade, v1.1 http://www.usb.org/developers/devclass_docs/DFU_1.1.pdf
UnicodeECN	USB Engineering Change Notice – UNICODE UTF-16LE for String Descriptors http://www.usb.org/developers/docs

I.III.III ISO

International Organization for Standardization, http://www.iso.org documents referenced:

Table I.3 Referenced ISO Documents

Ref	Title and Source
ISO-4217	Codes for the representation of currencies and funds
ISO/IEC 10646	Universal Character Set Encoding Specification

I.III.IV Other References

Table I.4 Referenced Miscellaneous Documents

Ref	Title and Source
PP0.8	PlusPower Specification, v0.8 http://www.poweredusb.org/pdf/PoweredUSB_v08g.pdf.
RFC 2781	UTF-16 Unofficial Specification
Unicode Standard	Unicode Standard Version 3.0 http://www.unicode.org

I.IV Document Conventions

I.IV.I Indicating Requirements, Recommendations, and Options

Terms and phrases in this document that indicate requirements, recommendations, and options in the protocol are used as defined in the IETF RFC 2119.

In summary:

Requirements:

To indicate requirements, this document uses "MUST", "MUST NOT", "REQUIRED", "SHALL", or "SHALL NOT".

Recommendations:

To indicate recommendations, this document uses "SHOULD", "SHOULD NOT", "RECOMMENDED".

Options:

To indicate **options**, this document uses "MAY" or "OPTIONAL".

I.IV.II Deprecated Functionality

Deprecated functionality is supported remains part of this protocol; however, GSA recommends that deprecated functionality not be used in new implementations.

In this document, deprecated functionality is identifiable by a pale **red** background and text detailing from what version the functionality was deprecated, and in which version the functionality originated. The following example (`field4` has been deprecated) shows how this convention is used in tables.

Table I.5 exampleReport Fields (Sheet 1 of 2)

Name	Value	Description
`field1`	`1-5`	Description text...

Table I.5 exampleReport Fields (Sheet 2 of 2)

Name	Value	Description
field2	0-255	Description text...
field3	0	Description text...
	1	Description text...
Deprecated in v1.1: v1.0.0		
field4		Description text...

NOTE:
Due to the nature of some tables within GDS protocols, such as report structure tables that give an overview of the byte and bit order, space constraints require that the version information be provided in a footnote for that table. See below for an example.

Table I.6 Report Structure Table Example[a]

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x00**							
Byte 1	Field1							
Byte 2	-	Field2	Field3	Field4	Field5	Field6	Field7	Field8

a. Deprecated in v1.1: v1.0.3 -- Byte 2 Bit 6: Field2

The report detail section for a deprecated report is treated similarly, with the section heading having the pale red background, such as in the example below.:

25.7 Old Report Name 0x00

Deprecated in v1.1: v1.0.0

I.IV.III Extended Functionality

A pale **green** background identifies extended functionality, along with text that identifies in what version the extension was added. The following example (`field4` is an extension) shows how this convention is used in tables:

Table I.7 exampleReport Fields (Sheet 1 of 2)

Name	Value	Description
field1	1-5	Description text...
field2	0-255	Description text...

Table I.7 exampleReport Fields (Sheet 2 of 2)

Name	Value	Description
field3	0	Description text...
	1	Description text...
Extension in v1.1		
field4		Description text...

NOTE:
Due to the nature of some tables within GDS protocols, such as report structure tables that give an overview of the byte and bit order, space constraints require that the explanatory text be provided in a footnote for that table. See below for an example.

Table I.8 Report Structure Table Example[a]

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x00**							
Byte 1	Field1							
Byte 2	-	Field2	Field3	Field4	Field5	Field6	Field7	Field8

a. Extension in v1.1 -- Byte 2 Bit 6: Field2

The report detail section added in an extension is treated similarly, with the section heading having the same pale green background, such as in the example below:

25.7 Report Name

Extension in v1.1

I.IV.IV Corrections and Clarifications

A pale **yellow** banner identifies content that has been corrected or clarified since the last released version, along with text that identifies in what version the changes were made. The following example shows how this convention is used, and indicates that corrections were made in v1.1 to content.

Note that correction banners and the associated inserted and deleted text is highlighted only in the mark-up PDF of released versions, and are provided only for changes made between the last released version and the current released version. Correction indicators are not carried forward from version to version.

Corrections in v1.1

Lorem ipsum dolor sit amet, consectetur ~~consectetur~~ adipiscing elit. Aliquam consectetur justo vel odio consequat rutrum. Morbi magna neque, blandit a dictum nec, vestibulum ac velit. Donec ultrices imperdiet mi, eget pharetra enim porttitor quis. Nam vestibulum massa eget augue consectetur condimentum tempus enim pellentesque.

I.IV.V Other Formatting Conventions

Plain text indicates specification text.

Blue text indicates an internal link or an external hyperlink to a URL.

Bold (other than in headings) or underlined text is used for emphasis, unless specifically indicated otherwise.

Italicized text (other than in headings) is used for terms being defined, unless specifically indicated otherwise.

`Courier New` font is used to indicate code or pseudo code.

I.V Document Organization

Correction in v1.3

Chapter **1**	Overview of purpose and benefits of the GDS note acceptor specification.
Chapter **2**	Hardware and software characteristics common to all GDS peripheral devices. Common reports are listed with references to the appropriate detail sections in chapters 3 and 4.
Chapter **3**, **4**	Command and event details, respectively. Both chapters are similarly organized. See Command and Event Detail Organization below.
Chapter **5**	Usage number and report ID allocation for the GDS note acceptor protocol, as well as high-level allocation for GDS peripheral device protocols in general. Report summary, in tabular format, note acceptor reports and report items, with corresponding details such as usage number and/or report ID.
Chapter **6**	USB identifier strings, such as vendor codes assigned by USB-IF, as well as other identifiers, including product and manufacturer identifiers.
Chapter **7**	Note Acceptor Metrics Data format. Format of data sent in the Metrics report.
Glossary	Definitions of terms.
Appendix **A**	CRC-32 checksum calculation, with C code examples, adapted for calculation time considerations and for space considerations.
Appendix **B**	HID report descriptor table for note acceptors, from a Cypress EZ-USB micro controller.

I.V.I Command and Event Detail Organization

Chapters 3 and 4 are similarly organized as follows:

1. Table listing the reports described in the chapter: report ID, usage number, text label, whether data is sent in the report, and page number where details appear.
2. Overview information, if any.
3. Report detail sections. Reports are listed in numerical order by report ID.
    a. Description of report.
    b. Usage examples or scenarios, if any.
    c. Usage rules, if any.
    d. Report structure table.
    e. Item description table, if applicable. If the only data sent in the report is the report ID, this table is not included.

Chapter 1

Introduction

1.1 Introduction

This is the specification for a USB note acceptor as defined by the GDS within the GSA. For the purposes of this document the terms “USB note acceptor”, “note acceptor”, “GDS device” and “device” may be used interchangeably. The aim of the GDS is to develop true plug ‘n’ play peripherals for the gaming environment which are already common place in the computer industry. Rather than use a derivative of RS232 it was decided to move the technology forward and adopt the USB standard.

This document specifies the complete set of functionality for the GDS Note Acceptor.

1.2 Note Acceptor Function Overview

Commands supported by the note acceptor are defined in Chapter 3. Supported events are defined in Chapter 4. Common commands and events are identified in Chapter 2.

The note acceptor on power-up must default to escrow mode, be disabled, run a diagnostic test, and report events as necessary.

The note acceptor stores accepted note data, even over power failure intervals, until the acknowledgment of the note data transaction is received from the host.

The interrupt service period, bInterval, is 100ms.

1.3 USB Compliance and Benefits

USB is currently available in three speeds:

- low speed: 1.5 Mbits/sec.
- full speed: 12Mbits/sec.
- high speed: 480Mbits/sec.

GDS peripherals MUST, at a minimum, support **full speed** and MUST comply with all published USB requirements.

Only one master exists on the USB bus and that is the host machine. All peripherals MUST be connected through hubs (up to 127 allowed but this is likely to be less than 10). For security reasons, it is undesirable to have a direct peripheral-to-peripheral communication path. Therefore, OTG (On- The-Go) extensions that allow peripherals to talk directly to one another MUST NOT be used; all communication MUST be via the host.

1.4 Note Acceptors as HID Class Devices

To remove the need for special manufacturer-specific device drivers, the note acceptor MUST be implemented as an HID—Human Interface Device—class device (see document reference DCDHID). This is the same class as mice, keyboards and joysticks but is a flexible enough to be used for general input/output data packets

where the data sizes are relatively small and the transfer rates low. HID implements usage pages and usage numbers to allow access to data in an abstracted fashion, regardless of the manufacturer or even the exact location within a report of where the data can be found.

This document assumes hardware and firmware are in place to support a HID class device within USB and so need only concern itself with HID commands and data formats for gaming. Product identification is also discussed.

1.5 Device Firmware Upgrade (DFU)

Correction in v1.3

GDS devices MUST use the Device Firmware Upgrade (DFU) standard, version 1.1, as defined by USB-IF, http://www.usb.org/developers/devclass_docs/DFU_1.1.pdf as the method of upgrading device firmware.

The use of the DFU class MUST NOT affect the behavior of the HID class.

Upon firmware upgrade the device MUST clear its internal memory including memory involving Transaction IDs.

The serial number exposed through the device descriptor in DFU and GDS modes MUST be identical.

Support for firmware download is REQUIRED; support for firmware upload is OPTIONAL.

Chapter 2

GDS Peripheral Common Design Characteristics

2.1 GDS Peripheral Common Design Characteristics

This section describes characteristics shared by all GDS peripheral devices.

2.2 Hardware: Communications

USB 2.0 defines low (1.5Mbits/sec.), full (12Mbits/sec.) and high (480 Mbits/sec.) speed communications. GSA devices MUST, at a minimum, support full speed and MUST comply with all USB requirements.

The USB specification defines a reliable means of communication at both hardware and software levels, including error detection and recovery. See document reference USB 2.0.

2.3 Hardware Power-Loss and Non-Reproducible Data Recovery

2.3.1 Non-Volatile Memory

Critical devices MUST implement non-volatile memory (NVM).

If battery-backup systems are implemented, the device MUST implement battery level circuitry to detect when the method of storage is no longer reliable.

If non-volatile memory fails, the non-volatile memory (NVM) failure event MUST be reported. See Page 29.

2.3.2 Power Fail Signaling

Devices MUST implement power fail detect circuitry and a power reserve capability when NVM is required.

2.4 Software: USB Classification

Devices MUST be classified as HID devices unless otherwise indicated.

2.5 Software: Communications Overview

A set of *standard* device requests are defined by USB. HID defines additional *class-specific* requests for managing reports. All device classes MUST implement GET/SET REPORT and GET/SET IDLE HID requests unless otherwise stated.

Under HID, all device communications MUST be encoded into a reporting structure as defined by Device Class Definition for Human Interface Devices. See document reference DCDHID.

The host sends a report to the peripheral to issue a command and where a peripheral wishes to notify the host of an event, it sends a report on the INTERRUPT IN endpoint as specified by the devices report descriptor. The report ID indicates the event type and the report data indicates any event data.

2.5.1 Information Format

The GDS protocol is a byte oriented protocol in that the commands, events, and data information are defined at the byte level. However, in some cases, the bytes are further defined to the bit level with each individual bit representing a specific function or indication. When bytes are defined to the bit level, the default value for the bits is zero (0). Additionally, bits not specifically assigned a function, i.e. the bit is unused, MUST be set to zero (0).

2.5.2 Report Delivery and Priority

If the host requires information from the peripheral, it requests a report. The report ID identifies the information to retrieve. Reports can be scheduled for delivery at regular intervals if desired. However, note that on power up, devices MUST handle messages in the following priority:

1. Power status event
2. Failure status event
3. Pending Transaction ID event
4. Any previously stored Transaction ID events that were in addition to the pending Transaction ID event

The design that follows, however, focuses on higher level primitives to assist those not familiar with the USB software layer.

Do not confuse the acknowledgment system defined by the USB protocol layer with the event acknowledgment system defined below.

The concept of a command, event and event acknowledgment are now defined.

2.5.3 Command

Commands specified in this document are sent from host to device. These commands are implemented as Set_Report (Feature) requests. See document reference USB 2.0. The following table identifies commands currently implemented as common reports (seeSection 5.2 and Section 5.3). These commands are used by multiple device classes.

Table 2.1 Common Commands

Report ID	Usage ID	Name	Data	
0x01	0x40	ACK	No	Page 13
0x02	0x41	Enable	No	Page 15
0x03	0x42	Disable	No	Page 15
0x04	0x43	Self Test	No	Page 15
0x05	0x44	Request GAT Report	No	Page 16
0x08	0x47	Calculate CRC	Yes	Page 16

2.5.4 Event

Events specified in this document are sent from device to host.

USB simulates the concept of an interrupt through regular polling intervals issued by the host's USB device driver. The interrupt service period is specified by the device via an endpoint descriptor. The interrupt interval chosen is based on the function of a device.

Events can be sent at regular intervals, when the device has something to send or when the host has requested that an event be sent.

An application level acknowledgment is required for some events. These events deal with currency related activities or fault activities. The Host must make sure that it has acted on them and not just received them before Acknowledging these events to the device.

The protection is needed such that, on power loss, these events are not lost but resent when power is restored and the host can decide if it had serviced the event prior to power loss or must do so now.

Status events need to be reported when a fault occurs and when it is cleared.

The following table identifies events currently implemented as common reports (seeSection 5.2 and Section 5.3). These events are used by multiple device classes.

Table 2.2 Common Events

Report ID	Usage	Event	Data	
N/A	USB Defined	Connection	No	Page 22
N/A	USB Defined	Disconnection	No	Page 23
0x06	0x45	Power Status	Yes	Page 23
0x07	0x46	GAT Data	Yes	Page 24
0x09	0x48	CRC Data	Yes	Page 26
0x0A	0x49	Device State	Yes	Page 26

2.5.5 Event Acknowledgement

Correction in v1.3

For Transaction ID (TID) events only, an event acknowledgment is required.

Critical events sent from a device and the host's acknowledgment command contain a Transaction ID.

Transaction IDs are one (1) byte in length and initially start at zero (0). Transaction IDs wrap to zero (0) after reaching the upper limit of 255.

The host increments its Transaction ID when the Transaction ID of the event matches the host's copy. The host sends a 0x01 ACK command (see Page 13) to the device only after it has serviced the event.

A device increments its Transaction ID when it receives an ACK from the host indicating the Transaction ID of the pending event.

If transaction numbers do not match, the host may send an ACK indicating the received TID after handling this mismatching event, may resynchronize Transaction IDs or may take any other action.

If Transaction ID events with TID x are received by the host while the host is busy processing an event with TID x, the host may choose to ignore these messages as they are duplicates of the event being processed.

The device resends the event when it receives an ACK with a wrong Transaction ID. However, if the device has just recovered from power loss and has detected that an event was waiting acknowledgement when power was lost, and has resent the event, the device discards the resent event as soon as the ACK is received. The host either has indicated that it had already serviced it or did just then. Regardless the Transaction ID is incremented and operation continues.

Transaction ID events can only be sent if there is no pending Transaction ID event. Devices must queue any un-sent and pending Transaction ID events.

If an event is unrecognized by the host, the host may ignore the event or take some other action.

If an ACK is not received within one (1) second then the event is resent to the host.

2.5.6 USB Enumeration Sequence

Correction in v1.3

After enumeration, the device MUST stay in the Disabled state. During this time the device MUST continuously monitor itself for any failure or fault events. During this time, only the 0x03 Disable command (see Page 15) is accepted by the device. Any other command MUST be ignored.

After the device receives its first Disable command from the host, it responds with the 0x0A Device State event (see Page 26)and then it reports any event messages in the following priority.

1. Power Status events, if external power is missing and the device needs external power to operate.
2. Perform self-test and report failure status before generating any other reports. While performing a self-test a device may NAK, at the USB level, any host commands.
3. Any pending credit or pending status events. These are defined as any note acceptor TID that was already placed into EndPoint 1 stack for transmission, but was not completely communicated to the host. A complete communication requires that the host acknowledge the event and also that the Transaction ID rules have been met whereby both the host and device Transaction IDs are in sync. These pending events must be transmitted to the host in their order of occurrence.
4. Any previously stored note acceptor TID events that were in addition to the pending TID event described in the above item #3.

NOTES:

1. A device MUST be ready to accept the Disable command from host within two (2) seconds after enumeration.
2. A host may be unaware that there are unreported TID events. Thus, a device MUST be prepared to receive and process commands from the host before all TID events have been reported to the host. For example, a device must be prepared to receive and process a Self Test command before all TID events have been reported.

2.5.7 Device and Host Interaction

Correction in v1.3

The device uses the USB Device Layer ACK and NAK handshake protocols to control the receipt of commands from the host. If the device NAKs a command, the host retries the transaction until the device responds with an ACK. The host may use any method to determine if a device has NAKed the host too long. Functional and Commanded STALL instructions MUST NOT be used when communicating via the GDS interface. Protocol STALL instructions MUST be used if the host sends an unsupported request. See the USB specification for information regarding Protocol STALL instructions.

NOTE:
For the purposes of this document, ignoring a command is interpreted to mean that the device MUST respond with a USB Device Layer ACK to complete the transaction, before discarding the command with no further response.

When the device acknowledges a command packet, the device accepts responsibility to complete the command. Commands MUST be completed within the allocated time-out periods. For the 0x08 Calculate CRC (see Page 16) and 0x04 Self Test (see Page 15) commands the host allows a maximum time-out of 20 seconds between the receipt of the ACK and the return of requested data on the Input End Point. This time-out is five (5) seconds for all other commands that operate during the disabled state and have an associated response to the host. The 0x03 Disable command (see Page 15) is an exception and MUST respond to the host within three bIntervals (see Section 1.2).

Care must be taken in the device firmware to ensure that a NAK_OUT is used without STATUS_IN being included because the host might interpret the ACK of the status packet as receipt of the command. The device firmware must also ensure that upon returning to an ACK_OUT state that the device also includes the STATUS_IN instruction to allow the command from the host to be recognized. To repeat: to stop receipt of commands from the host, the device must perform a NAK_OUT instruction; and to accept commands from the host, the device must perform an ACK_OUT_STATUS_IN instruction.

More information is available in section 5.3.2 and section 4.4 of the USB specification (see document reference USB 2.0).

2.5.8 Extensions to Command and Events

Correction in v1.3

The commands and events defined within this specification MUST only be extended in new releases of this specification. Any such extensions MUST be approved by the Gaming Standards Association. Additional features and functionality, beyond what is anticipated in this specification, MUST NOT be included an implementation. Inclusion of such features may cause unanticipated behaviors and interoperability issues.

For example, implementations must not use unassigned bits in device status reports to indicate the availability of implementation-specific features; and, implementations must not use commands or unassigned bits within commands to trigger implementation-specific behavior.

Chapter 3

Command Support

3.1 Command Support

The following note acceptor commands MUST be supported and implemented as HID Feature reports.

Table 3.1 Supported Note Acceptor Commands

Report ID	Usage ID	Name	Data	
0x01	0x40	ACK	No	Page 13
0x02	0x41	Enable	No	Page 15
0x03	0x42	Disable	No	Page 15
0x04	0x43	Self Test	No	Page 15
0x05	0x44	Request GAT Report	No	Page 16
0x08	0x47	Calculate CRC	Yes	Page 16
0x80	0x0210	Number of Note Data Entries	No	Page 17
0x81	0x0211	Read Note Table	No	Page 17
0x82	0x0212	Extend Timeout	No	Page 18
0x83	0x0213	Accept Note/Ticket	No	Page 18
0x84	0x0214	Return Note/Ticket	No	Page 18
Extension in v1.2				
0x8A	0x021A	Read Note Acceptor Metrics	No	Page 19

NOTE:
The following reports are *diagnostic commands*: Report 0x04 Self Test, Report 0x05 Request GAT Report, Report 0x08 Calculate CRC, Report 0x80 Number of Note Data Entries, and Report 0x81 Read Note Table.

3.2 Command Execution When Enabled/Disabled

The following table shows the device state a given command can be performed under. Any command sent to the note acceptor while it is in the wrong state MUST be ignored. Unless specified differently elsewhere, any command sent to the note acceptor while it is in the correct state MUST NOT be ignored.

Table 3.2 Command Execution When Enabled/Disabled

Command	Operation When Device Enabled	Operation When Device Disabled
0x02 Enable	Yes	Yes
0x03 Disable	Yes	Yes
0x04 Self Test	No	Yes
0x05 Request GAT Report	No	Yes
0x08 Calculate CRC	No	Yes
0x80 Number of Note Data Entries	No	Yes
0x81 Read Note Table	No	Yes
0x82 Extend Timeout	Yes	No
0x83 Accept Note/Ticket	Yes	No
0x84 Return Note/Ticket	Yes	No
Extension in v1.2		
0x8A Read Note Acceptor Metrics	No	Yes

3.3 Report 0x01 ACK

This is a pseudo-command as it is sent in response to a Transaction ID event from the note acceptor as part of the handshake sequence. This command is used to confirm critical events such as Note Validated, Ticket Validated, Note/Ticket Status and Stacker Status events.

The Note Validated event, Ticket Validated event, Note/Ticket Status event and Stacker Status event share the same Transaction ID sequence, and are referred to as note acceptor Transaction ID, or TID, events.

A Transaction ID stamp is used to ensure that every note acceptor TID event is properly handled by the host and only acknowledged events are removed from the note acceptor queue. If the device does not receive an ACK from the host within one (1) second then the event MUST be resent to the host.

The Transaction ID protocols are only used with TIDs. If power is removed before all of the note acceptor TID events have been completely communicated with the host, they MUST be re-issued by the acceptor when power is restored and enabled. Therefore, any un-sent or pending note acceptor TIDs MUST be buffered in NVM.

3.3.1 Transaction ID Rules

Correction in v1.3

1. The Transaction ID sequence is common to events that contain a Transaction ID. The Transaction ID MUST be incremented sequentially each time one of these events occurs and is acknowledged. For example, a 0x86 Note Validated event (see Page 31) with a Transaction ID of 100 would be followed by a 0x88 Note/Ticket Status event (see Page 32) with a Transaction ID of 101.
2. Transaction IDs MUST increment through the range 0 to 255.
3. Transaction IDs MUST NOT be cleared from the note acceptor stack except:
    a. After the host has acknowledged the event and the Transaction ID rules have been met whereby both the host and device Transaction IDs are in sync.
    b. Upon a firmware upgrade of the device (DFU). In this case all Transaction IDs MUST be cleared.
4. The current Transaction ID MUST be stored in the device's NVM so as to provide a reference number in the event of a power interruption.
5. If the device does not receive an acknowledge from the host within one (1) second after sending a critical event, the device MUST retry sending the event every one (1) second until the host properly acknowledges the event and the Transaction IDs are in sync.
6. While waiting for an ACK, a device MUST NOT NAK, at the USB level, a command from the host.
7. When the Resync bit is set in an ACK command, the device MUST reset the current Transaction ID to the value specified by the host in the ACK command. The pending event MUST then be retried using the new the Transaction ID. The Transaction ID MUST be incremented sequentially from that new Transaction ID. For example, if an event is sent with Transaction ID 10 and the host resyncs the Transaction ID to 20, the event will be retried with Transaction ID 20 and the next event will be sent with Transaction ID 21.

Table 3.3 0x01 ACK Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x01**							
Byte 1	-	-	-	-	-	-	-	Resync
Byte 2	Transaction ID							

Table 3.4 0x01 Field Descriptions

Name	Value	Description
Resync	0	The device MUST NOT re-sync its Transaction ID. This is an acknowledgement.
	1	The device MUST re-sync its Transaction ID and resend the pending report.
Transaction ID	0 to 0xFF	The confirmed or new Transaction ID. SeeSection 3.3.1.

3.4 Report 0x02 Enable

The device MUST activate any inputs and/or enable any outputs. If the device has a fault then it MUST stay disabled and report the fault to the host.

The host issues this command to allow notes to be drawn into the note acceptor for the purposes of validation.

Animation lamps are enabled.

If the device has no faults, after enabling itself, the device MUST send the 0x0A Device State event with the Enable bit set (see Page 26).

Table 3.5 0x02 Enable Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x02**							

3.5 Report 0x03 Disable

Correction in v1.3

The device MUST deactivate any inputs and/or disable any outputs. The host issues this command when it does not want to allow notes/tickets to be drawn into the note acceptor. If a note/ticket is in escrow when this command is received, the device MUST return the note/ticket immediately after disabling itself; the device MUST NOT wait for the timeout period for the note/ticket to expire before returning the note/ticket.

The host MUST send a Disable command before issuing any diagnostic commands (defined on Page 61).

Animation lamps are disabled; however, any lamps that indicate that the device is out of service are not affected.

When the device is in the enabled state and receives a Disable command, the device MUST immediately disable itself and respond with the 0x0A Device State event (see Page 26).

Table 3.6 0x03 Disable Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x03**							

3.6 Report 0x04 Self Test

The device MUST initiate a self-test sequence when instructed by the host. When the host requests a self-test, the device MUST respond with a 0x85 Failure Status event (see Page 29). The device MUST complete all tests before sending the Failure Status event to the host. Multiple failures MUST be presented in the final single

Failure Status event, with the exception of multiple 'Other' failures, which MUST be reported in separate Failure Status events.

Table 3.7 0x04 Self Test Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x04**							
Byte 1	-	-	-	-	-	-	-	NVM

Table 3.8 0x04 Field Descriptions

Name	Value	Description
NVM	0	Perform self-test.
	1	Non-volatile Memory (NVM), previously queued Transaction ID events and Transaction ID sequence number, MUST be cleared before the self-test is performed.

3.7 Report 0x05 Request GAT Report

The host sends this command to request information from the note acceptor via a 0x07 GAT Data event (see Page 24). The device is not required to monitor itself for fault conditions during the performance of this command. The 0x07 GAT Data response MUST be in ASCII format. The following characters MUST NOT be used in the response data: '<' or '/>'. The note acceptor does not output in XML.

At this time, the serial and network-based GAT protocols, such as GAT v3.50, G2S v1.1, and S2S 1.5, do not require the use of this command. Authentication of peripheral devices is performed using the 32-bit CRC algorithm. This command is simply a placeholder. The GAT data in the response is manufacturer-specific in terms of length and content. The response MAY include no data bytes.

Table 3.9 0x05 Request GAT Report Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x05**							

3.8 Report 0x08 Calculate CRC

The host can request a 32-bit checksum from the code space within the note acceptor's ROM memory, excluding the USB Identification Strings (see Page 50). For additional accuracy and security, a 32 bit seed is sent to the device as a parameter of this command. The host may choose any seed value and compare the result with a look-up table or a similar calculation based on an exact reference copy of the code to be verified.

A checksum report is returned in the 0x09 CRC Data event (see Page 26) with the result.

While calculating the CRC the device is not required to receive any additional commands. The device is permitted to NAK_OUT all requests from the host while performing the CRC calculation. The device is also not required to monitor itself for fault conditions during the performance of this command.

See Page 63 for the algorithm used.

Table 3.10 0x08 Calculate CRC Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x08**							
Byte 1	Seed 0							
Byte 2	Seed 1							
Byte 3	Seed 2							
Byte 4	Seed 3							

Table 3.11 0x08 Field Descriptions

Name	Value	Description
Seed	0 to 255	The starting seed for the CRC-32 calculation. Seed 0 is the LSB.

3.9 Report 0x80 Number of Note Data Entries

The host issues this command to first determine how many note data entries are expected to be read or written when an upgrade or 0x81 Read Note Table command (see Page 17) is executed.

Table 3.12 0x80 Number of Note Data Entries Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x80**							

3.10 Report 0x81 Read Note Table

The host issues this command to get the list of notes supported by the note acceptor. If note data is upgraded, the host must re-issue this command as the number of entries and/or value associated with a given Note ID (see Page 28) may have changed.

Table 3.13 0x81 Read Note Table Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x81**							

3.11 Report 0x82 Extend Timeout

The host issues this command in response to either 0x86 Note Validated event (see Page 31) or 0x87 Ticket Validated event (see Page 31) when it needs to retain a note/ticket in escrow for a longer period. The Extend Timeout command resets the timeout period to five (5) seconds.

Table 3.14 0x82 Extend Timeout Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x82**							

3.12 Report 0x83 Accept Note/Ticket

The host issues this command in response to either 0x86 Note Validated event (see Page 31) or 0x87 Ticket Validated event (see Page 31) when it wants to accept and stack a note/ticket. The host must not credit the note/ticket value until a 0x88 Note/Ticket Status event (see Page 32) is received from the device with the Accepted bit set.

The device MUST automatically return the note/ticket if it does not receive a command to accept or return the note/ticket from the host within five (5) seconds from sending the 0x86 Note Validated or 0x87 Ticket Validated event and the timeout has not been extended. A 0x88 Note/Ticket Status event, (see Page 32) MUST be issued by the device if a note/ticket is returned due to lack of host response.

The host MAY extend the timeout period by issuing 0x82 Extend Timeout command (see Page 18) before the device's timeout period expires. The host MAY continue to extend the device's timeout period indefinitely.

Table 3.15 0x83 Accept Note/Ticket Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x83**							

3.13 Report 0x84 Return Note/Ticket

Correction in v1.3

The host issues this command in response to either 0x86 Note Validated event (see Page 31) or 0x87 Ticket Validated event (see Page 31) when it wants to return the note/ticket. The device MUST automatically return the note/ticket if it does not receive a command to accept or return the note/ticket from the host within five (5) seconds from sending the 0x86 Note Validated or 0x87 Ticket Validated event and the timeout has not been extended. After returning the note/ticket, the device MUST generate the 0x88 Note/Ticket Status event (see Page 32) with the Returned bit set for confirmation. See Section 4.17.2.1, Note Acceptor Automatic Reject, for special requirements that apply if the host instructed the note acceptor to reject the note/ticket but the note/ticket was not returned before a power cycle or system reset occurred.

The host MAY extend the timeout period by issuing 0x82 Extend Timeout command (see Page 18) before the device's timeout period expires. The host MAY continue to extend the device's timeout period indefinitely.

Table 3.16 0x84 Return Note/Ticket Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x84**							

3.14 Report 0x8A Read Note Acceptor Metrics

Extension in v1.2

The host issues this command to request the note acceptor's capability metrics, such as types of barcodes supported and UTF-16 support. The note acceptor sends the 0x8A Read Note Acceptor Metrics event (Page 35) in response.

Table 3.17 0x8A Read Note Acceptor Metrics Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x8A**							

Chapter 4

Event Support

4.1 Event Support

The following note acceptor events MUST be supported. SeeSection 2.5.2 for message priority.

Table 4.1 Supported Note Acceptor Events

Report ID	Usage ID	Event	Data	
N/A	USB Defined	Connection	No	Page 22
N/A	USB Defined	Disconnection	No	Page 23
0x06	0x45	Power Status	Yes	Page 23
0x07	0x46	GAT Data	Yes	Page 24
0x09	0x48	CRC Data	Yes	Page 26
0x0A	0x49	Device State	Yes	Page 26
0x80	0x0210	Number of Note Data Entries	Yes	Page 27
0x81	0x0211	Read Note Table	Yes	Page 28
0x85	0x0215	Failure Status	Yes	Page 29
0x86	0x0216	Note Validated	Yes	Page 31
0x87	0x0217	Ticket Validated	Yes	Page 31
0x88	0x0218	Note/Ticket Status	Yes	Page 32
0x89	0x0219	Stacker Status	Yes	Page 34
Extension in v1.2				
0x8A	0x021A	Read Note Acceptor Metrics	Yes	Page 35
0x8B	0x021B	UTF Ticket Validated	Yes	Page 36

The GDS devices are event driven and therefore the detection of an error as well as the clearance of it MUST be reported. This guarantees that the host is informed of the state of a given device.

4.2 Connection/Disconnection of Communications

The USB specification handles connection/disconnection of a device. See document reference USB 2.0.

4.3 Connection/Disconnection of Power

Devices may require additional power to that supplied by the USB and if so, the standard USB configuration descriptor is used to inform the host of this requirement.

On connecting the device USB cable to the host, the host detects the device, although the device may not be fully operational as in the case when the device requires external power and that power is not connected.

Devices that require external power must report a 0x06 Power Status event (see Page 23) per the rules inSection 4.4.1. The host, must be notified whenever a loss or reapplication of external power occurs. In the case where the USB communications section of the device is powered by the USB +5 power, the host will be notified by the device sending a Power Status event with the appropriate status bits set. In the case where the USB communications section of the device is powered by the external power, the host will be notified by virtue of the fact that the device will drop out of the addressing scheme when the external power is removed and will become re-enumerated and re-addressed when the external power is reapplied.

If the device does not require external power, the power status event need not be sent.

If the device requires external power, and if external power is disconnected, the device MUST disable itself. The device MUST remain disabled until enabled by the host.

4.4 Report 0x06 Power Status

Correction in v1.3

The presence or absence of external power is indicated in this report with the Ext. Power flag. The report MUST only be generated by peripherals which need external power to operate.

Another flag, Need Reset, is used to indicate whether a device MUST be reset explicitly by the host if external power is connected after a disconnection state. Most GDS devices will be able to generate their own reset on connection of power.

4.4.1 Rules for External Power Status

Correction in v1.3

If the USB communications section of the GDS device is still capable of communicating with the host:

1. and it has no external power present or the external power is insufficient for the device to operate, if the GDS device is able to enumerate, after receiving a “Disable” command from the host, it MUST generate a Device State Report 0x0A to the host followed by the Power Status Report 0x06. While external power remains unavailable, the GDS device MUST respond to any subsequent commands from the host by generating a Power Status Report 0x06.
2. and external power is lost or becomes insufficient for the device to operate, the GDS device MUST respond to all commands from the host by generating a Power Status Report 0x06.
3. and external power is lost or becomes insufficient for the device to operate, the GDS device MUST abort any diagnostic process currently running as a result of an initiating command (such as Calculate CRC Report 0x08 or others) and MUST generate a Power Status Report 0x06.

4. and external power is lost or becomes insufficient for the device to operate, the GDS device MAY monitor itself for additional conditions but MUST NOT generate any command except Power Reset Report 0x06 until external power has been connected and becomes sufficient for the device to operate.

In all cases, when external power is restored or becomes sufficient for the device to operate, the GDS device MUST generate a Power Status Report 0x06 indicating external power has been connected.

Table 4.2 0x06 Power Status Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x06**							
Byte 1							Need Reset	Ext. Power

Table 4.3 0x06 Field Descriptions

Name	Value	Description
Ext. Power	0	External power is NOT connected or is NOT sufficient for the device to operate.
	1	External power is connected and is sufficient for the device to operate.
Need Reset	0	The device does not have to be explicitly reset by the host if external power is connected after a disconnection state—that is, after the Ext. Power bit transitions from 0 to 1.
	1	The device MUST be reset explicitly by the host if external power is connected after a disconnection state—that is, after the Ext. Power bit transitions from 0 to 1. Note that Reset is an intrinsic USB function.

4.5 Report 0x07 GAT Data

This event is sent in response to the 0x05 Request GAT Data Report, on Page 16. A block of data is returned containing various identification, diagnostic and auditing data in a format specific to each manufacturer and product.

All data MUST be in ASCII format. New lines are indicated with <carriage return><line feed> control characters (or decimal values 13 and 10).

XML tags MUST NOT be included in the data—this is done outside the peripheral. The following characters are not allowed in the GAT data:

/ (slash), decimal value 47

< (less than), decimal value 60

> (greater than), decimal value 62

If the total Data size exceeds 60 bytes, the 0x07 GAT Data report MUST be broken into a number of sequential reports.

Data up to 60 bytes in size MUST be sent in a single report with the Index set to one (1) and the Size between 0 – 60. If the GAT reply is longer than this then the data MUST be split into sequential packets with all packets having Size set to 61 bytes followed by a terminating packet which MUST have Size set to less than 61 bytes. In this way the host knows when all data has been received. If the last data packet contains 61 bytes exactly then a null packet MUST be sent as a terminator. This packet will have a Size of zero (0). Index numbers MUST start at one (1) and increment by one (1) with each packet sent. The host can stitch all the GAT packets together in the Index number sequence to reproduce the complete GAT report.

Request GAT Report	**GAT Data**

`TX:`<none>	`RX:[ Index  ]`
	`[ Size  ]`
	`[ Byte 1 ]`
	…
	`[ Byte 61 ]`

Table 4.4 0x07 GAT Data Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x07**							
Byte 1	Index							
Byte 2	Size							
Byte 3	Data 1							
...								
Byte 63	Data 61							

Table 4.5 0x07 Field Descriptions

Name	Value	Description
Index	1 to 255	Packet sequence counter for multi-packet reports.
Size	0 to 61	Last packet MUST contain less than 61 characters.
Data	10, 13, 32 to 126	ASCII characters (manufacturer specific), except the following which are not allowed: / (slash), decimal value 47 < (less than), decimal value 60 > (greater than), decimal value 62

4.6 Report 0x09 CRC Data

This event is sent in response to the 0x08 Calculate CRC command (see Page 16). The CRC checksum is calculated using a seed value provided in the 0x08 Calculate CRC command (see Page 16).

Calculate CRC	CRC Data
TX: [Seed 0 - LSB] [Seed 1] [Seed 2] [Seed 3 - MSB]	RX: [Result 0 - LSB] [Result 1] [Result 2] [Result 3 - MSB]

4.6.1 CRC Rules

1. When a CRC request is acknowledged by the device, the CRC Data event containing the 32 bit CRC checksum result MUST be available on the Input End Point within 20 seconds.
2. The checksum address range is pre-determined by the device firmware and MUST cover all re-programmable areas of memory, except for USB Identification Strings.

Table 4.6 0x09 CRC Data Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x09**							
Byte 1	Result 0							
Byte 2	Result 1							
Byte 3	Result 2							
Byte 4	Result 3							

Table 4.7 0x09 Field Descriptions

Name	Value	Description
Result	0 to 255	Result of the CRC-32 calculation. Result 0 is the LSB.

4.7 Report 0x0A Device State

This event is sent in response to 0x02 Enable (see Page 15) and 0x03 Disable (see Page 15) commands from the host, and indicates whether the device is enabled or disabled. A device MUST NOT report both Enabled and Disabled. The following scenarios A through C are examples of how this event is implemented.

Scenario A:

1. Host issues Enable.
2. Device enables itself and thus, also sends Device State Event.

Scenario B:

1. Device has fault.

2. Device disables itself.
3. Host issues Enable command.
4. Device stays disabled.
5. Since there is an active fault/failure, the Device must send fault/failure report. There is no need to send the Device State Event.

Scenario C, Special situation at Boot, after enumeration:

1. Device boots in Disabled State.
2. Device has Credits Pending, no faults or failures.
3. Host issues Disable command.
4. Device responds with DeviceState event with Disable bit set.
5. Device communicates all pending credits to the host.
6. Device remains disabled.

Table 4.8 0x0A Device State Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x0A**							
Byte 1	-	-	-	-	-	-	Disable	Enable

Table 4.9 0x0A Field Descriptions

Name	Value	Description
Disable	0	--
	1	Device disabled.
Enable	0	--
	1	Device enabled.

4.8 Report 0x80 Number of Note Data Entries

This event is sent in response to 0x80 Number of Note Data Entries command. It MUST contain the number of note entries the device is capable of validating. It MUST include all the notes that can be validated, even if the note acceptor is configured (via dip switches or similar mechanism) not to accept some of them at that time.

Table 4.10 0x80 Number of Note Data Entries Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x80**							
Byte 1	Number							

Table 4.11 0x80 Field Descriptions

Name	Value	Description
Number	0 to 255	Number of notes in table.

4.9 Report 0x81 Read Note Table

Correction in v1.3

This event is sent in response to the 0x81 Read Note Table command. A separate Read Note Table descriptor MUST be returned for each note supported by the note acceptor, including the notes that a note acceptor can validate but does not accept at that time due to a particular dip switch bank (or similar mechanism) configuration. When a note is inserted, a Note ID is specified in the 0x86 Note Validated Report that matches the Note ID returned in one of the 0x81 Read Note Table reports. This allows the host to efficiently look up the details of the note and does not require that the 0x81 Read Note Table Reports are received in any particular order.

The device is not required to monitor itself for fault conditions during the performance of this command.

Table 4.12 0x81 Read Note Table Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x81**							
Byte 1	Note ID							
Byte 2	Currency							
Byte 3								
Byte 4								
Byte 5	Value							
Byte 6								
Byte 7	Sign	Scalar						
Byte 8	Version							

Table 4.13 0x81 Field Descriptions (Sheet 1 of 2)

Name	Value	Description
Note ID	1 to 255	Note ID in Note Table.
Currency	AAA to ZZZ	The currency of the note. MUST conform to ISO-4217 (see document reference ISO-4217). Three characters in length, allowable characters: A-Z (ASCII 65-90). The first character of the currency code MUST be in Byte 2, the second character in Byte 3, and the third character in Byte 4.

Table 4.13 0x81 Field Descriptions (Sheet 2 of 2)

Name	Value	Description
Value	0 to 65535	Value of the note in arbitrary units. **Byte 5** is **LSB**. The scalar is applied to this value to produce a value indicative of that currency.
Scalar	0 to 127	The scalar specifies the indices in the equation 10 to the power of *scalar*. This result MUST be applied to the value of a note to obtain the true currency value in the primary unit of the currency.
Sign	0	Indicates a value of 10 to the power of –*scalar*.
	1	Indicates a value of 10 to the power of +*scalar*.
Version	0	No versions of this note exists.
	1 to 255	Indicates the version for a particular note denomination.

4.10 Report 0x85 Failure Status

Correction in v1.3

A Failure Status reports any faults with the peripheral device.

The flags are meant to be generic enough to be used by any manufacturer of note acceptor equipment. Manufacturers MAY use Byte 2 to provide a more specific fault code (as defined by GSA).

When instructed by the host, the device MUST perform a self-test and then send a Failure Status report to the host reporting the current status of the device.

During normal operation, the device MUST also continually check components that may lead to faulty operation. When doing so, if a fault condition is detected, a Failure Status report MUST be sent to the host unless specified differently elsewhere.

4.10.1 Failure Status Rules

Correction in v1.3

1. If any of the bits are set in this register then there is a fault with the note acceptor.
2. The device MUST self-disable until re-enabled by host.
3. If a fault clears, it MUST be reported to the host as being cleared.
4. A 0x85 Failure Status report MUST be sent to the host after receiving a 0x04 Self Test command (see Page 15), after receiving a 0x02 Enable command (see Page 15) if a fault exists, and as soon as a new fault condition is detected.
5. If the Other bit is set, a diagnostic code MUST be provided.
6. The failure status event assumes the highest priority for event transmission to the host. All Transaction ID events MUST wait until the failure is cleared and device is enabled.

7. If the NVM fault self clears then the NVM Failure event is sent to the host with this bit cleared. This could occur, for example, if the NVM fault was caused by static or RFI (Radio Frequency Interference).
8. A 0x85 Failure Status report with all bits set to 0, including the Diagnostics field, indicates that there are no more active failures and that the device is ready to report events and/or accept commands.

Table 4.14 0x85 Failure Status Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x85**							
Byte 1	Other	-	-	NVM	Comp	Opt	Mech	Firmware
Byte 2	Diagnostics							

Table 4.15 0x85 Field Descriptions

Name	Value	Description
Firmware	0	Firmware is OK.
	1	Firmware has failed.
Mech	0	Mechanical components are OK.
	1	Mechanical components have failed.
Opt	0	Optical components are OK.
	1	Optical components have failed.
Comp	0	Additional components are OK.
	1	One or more components have failed.
NVM	0	NVM is OK.
	1	A fault has been detected in the non-volatile memory.
Other	0	Indicates that the Diagnostic code in the Diagnostic byte is cleared. If, however, the Diagnostic code is set to 0 (zero), this indicates that all failures are cleared.
	1	A Diagnostic code is provided in the Diagnostics byte.
Diagnostics	0	No fault Information.
	1 - 255	See Table 4.16 below.

Table 4.16 0x85 Diagnostic Codes

Diagnostic Code	Description
1	Unable to continue self-test.
2 - 254	Reserved for future use.
255	Unknown error.

4.11 Report 0x86 Note Validated

This event MUST be sent when the note acceptor has verified that a note inserted has been validated.

The note acceptor MUST wait for a response from the host for five (5) seconds before the note is automatically returned. The host MAY issue a 0x82 Extend Timeout command (see Page 18) in response to this event before issuing 0x83 Accept Note/Ticket (see Page 18) or 0x84 Return Note/Ticket (see Page 18) commands.

Table 4.17 0x86 Note Validated Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x86**							
Byte 1	Transaction ID							
Byte 2	Note ID							

Table 4.18 0x86 Field Descriptions

Name	Value	Description
Transaction ID	0 to 255	Transaction ID of note event. Each event MUST be acknowledged by the host before it is removed from the stack. Events MUST be protected from power failure.
Note ID	1 to 255	For the host to interpret this value, the host MUST download the note table. The Note ID specified here MUST match the Note ID returned in one of the 0x81 Read Note Table reports.

4.12 Report 0x87 Ticket Validated

This event MUST be sent when the note acceptor device has verified that a ticket inserted has been validated, and it contains the bar code details. This event can only be used to report ASCII bar codes. Bar codes encoded using UTF character sets MUST be reported using event 0x8B UTF Ticket Validated (see Page 36). For ticket format, see document reference BCTS.

The note acceptor MUST wait for a response from the host for five (5) seconds before the ticket is automatically returned. The host MAY issue a 0x82 Extend Timeout command (see Page 18) in response to this event before issuing 0x83 Accept Note/Ticket (see Page 18) or 0x84 Return Note/Ticket (see Page 18) commands.

It is important to note that the report is 27 Bytes in length and that gives the bar code a maximum of 24 bytes.

Table 4.19 0x87 Ticket Validated Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x87**							
Byte 1	Transaction ID							

Table 4.19 0x87 Ticket Validated Structure

Bit	7	6	5	4	3	2	1	0
Byte 2	Length							
Byte 3	Code							
...								
Byte 26								

Table 4.20 0x87 Field Descriptions

Name	Value	Description
Transaction ID	0 to 255	Transaction ID of note event. Each event MUST be acknowledged by the host before it is removed from the stack. Events MUST be protected from power failure.
Length	1 to 24	The number of bytes taken up by the bar code in the following fixed-length 24-byte packet.
Code	0 to 255	A 24-byte ASCII string, right-padded with nulls.

4.13 Report 0x88 Note/Ticket Status

Correction in v1.3

This event MUST be sent in response to a host command to accept or return a note or when a note jam or cheat event is detected. This status MUST also be sent if the device powers up with a pending note/ticket, not accepted or rejected, that gets returned by the device. The rejected, returned and accepted status bits are mutually exclusive. For ticket format, see document reference BCTS.

Table 4.21 0x88 Note/Ticket Status Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x88**							
Byte 1	Transaction ID							
Byte 2	Jam	Cheat	-	Note Path Clear	Removed	Rejected	Returned	Accepted

Table 4.22 0x88 Field Descriptions (Sheet 1 of 3)

Name	Value	Description
Transaction ID	0 to 255	Transaction ID of note event. Each event MUST be acknowledged by the host before it is removed from the stack. Events MUST be protected from power failure.

Table 4.22 0x88 Field Descriptions (Sheet 2 of 3)

Name	Value	Description
Accepted	0	-
	1	The note/ticket has been accepted. The device sets this bit and sends the event in response to 0x83 Accept Note/ Ticket (see Page 18) command; however, the event MUST only be sent after the note passes a point of no return. The host gives credit for the previously validated note/ticket on this bit being sent. This event bit MUST be cleared after the event has been acknowledged by the host.
Returned	0	-
	1	A note/ticket has been returned to a position where the player can remove it. The device sets this bit and sends the event (a) in response to a 0x84 Return Note/Ticket (see Page 18) command or (b) if the device does not receive a command to accept/return the note/ticket within the timeout period, or (c) if a device error forces the note to be returned before receiving a response from the host. The device MUST NOT send multiple messages, for the same Returned event, when the note is not removed from its Returned position. The host waits for this event in response to a Return Note/Ticket command. This event bit MUST be cleared after the event has been acknowledged by the host.
Rejected	0	-
	1	A note/ticket inserted could not be verified. The device sets this bit and sends the event in response to an inserted note that failed verification. However, the message MUST only be sent after the note has reached a position where the player can remove it, even if it is left in that position. The device MUST NOT send multiple messages for the same Rejected event when the note is not removed from its Rejected position. This event bit MUST be cleared after the event has been acknowledged by the host.
Removed	0	-
	1	This bit, when set, indicates that a Rejected or Returned note or ticket was removed by the player from the device. This event bit MUST be cleared after the event has been acknowledged by the host. If a note acceptor device is not capable of detecting a "note/ticket removed" event, the device is still expected to send report 0x88 with byte 2, bit 3 set to 1. This bit can be set to 1 in the very same report that had the Returned or Rejected bit set to 1, or it may be set in a message following the one with the Returned or Rejected bit set to 1.

Table 4.22 0x88 Field Descriptions (Sheet 3 of 3)

Name	Value	Description
Note Path Clear	0	-
	1	This bit is set if the note or ticket that is in process is no longer present after a power cycle. This event bit MUST be cleared after the event has been acknowledged by the host.
Cheat	0	-
	1	A note/ticket cheat event has been detected. The device sets this bit and sends this event when a cheat event is detected. This event bit MUST be cleared after the event has been acknowledged by the host.
Jam	0	There are currently no note/ticket jams. The device sets this bit and sends the event when the jam has been cleared. The event is sent on a transition from 1 to 0 or as otherwise required.
	1	A note/ticket is jammed in the acceptance path. The device sets this bit and sends this event on a note/ticket jam. The event is sent on a transition from 0 to 1 or as otherwise required.

4.14 Report 0x89 Stacker Status

At all times, the device MUST assess these conditions and send this event to the host indicating the respective states.

Table 4.23 0x89 Stacker Status Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x89**							
Byte 1	Transaction ID							
Byte 2	Fault	-	-	-	-	Jam	Full	Disconnect

Table 4.24 0x89 Field Descriptions (Sheet 1 of 2)

Name	Value	Description
Transaction ID	0 to 255	Transaction ID of note event. Each event MUST be acknowledged by the host before it is removed from the stack. Events MUST be protected from power failure.
Disconnect	0	The stacker is connected.
	1	The stacker is disconnected. The note acceptor MUST disable itself and report the event. The fault is cleared by connecting the stacker but the device MUST remain disabled until enabled by the host.

Table 4.24 0x89 Field Descriptions (Sheet 2 of 2)

Name	Value	Description
Full	0	The stacker is not full.
	1	The stacker is full. The device MUST disable itself and report the event. The device clears this bit and sends the event when the stacker is emptied. The device MUST remain disabled until enabled by the host.
Jam	0	The stacker has no jams.
	1	The stacker has a jam. The device sets this bit and sends the event when a note/ticket jam occurs. The note acceptor MUST automatically disable itself. The fault is cleared by removing the jam but the device MUST remain disabled until the host enables it.
Fault	0	The stacker has no fault.
	1	The stacker has a fault. The note acceptor MUST automatically disable itself. The fault is cleared by replacing the stacker but the device MUST remain disabled until the host enables it.

4.15 Report 0x8A Read Note Acceptor Metrics

Extension in v1.2

The note acceptor sends this report in response to the 0x8A Read Note Acceptor Metrics command (Page 19), to inform the host of the UTF-16 support, and barcode types supported.

The format of the data in this command is defined in Chapter 7 (Page 55).

If the total Data size exceeds 60 bytes it MUST be broken into a number of sequential reports. Data up to 60 bytes in size MUST be sent in a single report with the Index set to one (1) and the Size between 0 – 60. If the Data is longer than this then it MUST be split into sequential packets with all packets having a Size of 61 bytes followed by a terminating packet, which MUST have a Size of less than 61 bytes. In this way the host knows when all data has been received. If the last data packet contains 61 bytes exactly then a null packet MUST be sent as a terminator. This packet will have a Size of zero (0). Index numbers MUST start at one (1) and increment by one (1) with each packet sent. The host can stitch all the Data packets together in the Index number sequence to reproduce the complete Data report.

The first report will have Index = 1. If more reports are needed to return all the Field Data then they are indexed 2, 3, 4…etc.

The Size byte indicates how many bytes are used within the fixed size report structure as the last report may not need all the available space.

Table 4.25 0x8A Read Note Acceptor Metrics Response Structure (Sheet 1 of 2)

Bit	7	6	5	4	3	2	1	0
Byte 0	**0x8A**							

Table 4.25 0x8A Read Note Acceptor Metrics Response Structure (Sheet 2 of 2)

Bit	7	6	5	4	3	2	1	0
Byte 1	Index							
Byte 2	Size							
Byte 3	Data 1							
...	...							
Byte 63	Data 61							

Table 4.26 Read Note Acceptor Metrics Response Field Descriptions

Name	Value	Description
Index	1 to 255	Packet sequence counter for multi-packet reports.
Size	0 to 61	Number of bytes of data to follow. Last packet MUST contain less than 61 characters.
Data	32 to 126	ASCII characters as defined in the Metrics Command Data Format section. See Page 55.

4.16 Report 0x8B UTF Ticket Validated

Extension in v1.2

This event is sent when the note acceptor device has verified that a ticket has been validated and contains the bar code details encoded using UTF character sets. This event can be used to report UTF or ASCII bar codes. Bar codes encoded using the ASCII character set MAY be reported using event 0x87 Ticket Validated (see Page 31). For ticket format, see document reference BCTS.

The note acceptor waits for a response from the host within five (5) seconds before the ticket is automatically returned. The host may issue a 0x82 Extend Timeout command (see Page 18) in response to this event before issuing 0x83 Accept Note/Ticket (see Page 18) or 0x84 Return Note/Ticket (see Page 18) commands.

Bar code information can be reported as ASCII or UTF-16LE data. The encoding is determined by examining the metrics report for the note acceptor. The metrics report is available in report 0x8A Read Note Acceptor Metrics (see Page 35). When the metrics report indicates that UTF is supported, the bar code information is encoded using UTF-16LE. Otherwise, the bar code information is encoded using ASCII.

If the total Data size exceeds 59 bytes it MUST be broken into a number of sequential reports. Data up to 59 bytes in size MUST be sent in a single report with the Index set to one (1) and the Size between 0 – 59. If the Data is longer than this then it MUST be split into sequential packets with all packets having a Size of 60 bytes followed by a terminating packet, which MUST have a size of less than 60 bytes. In this way the host knows when all data has been received. If the last data packet contains 60 bytes exactly then a null packet MUST be sent as a terminator. This packet will have a Size of zero (0). Index numbers MUST start at one (1) and increment by one (1) with each packet sent. The host can stitch all the Data packets together in the Index number sequence to reproduce the complete Data report.

Table 4.27 0x87 Ticket Validated Structure

Bit	7	6	5	4	3	2	1	0
Byte 0	0x8B							
Byte 1	Transaction ID							
Byte 2	Index							
Byte 3	Size							
Byte 4	Data 1							
…	…							
Byte 63	Data 60							

Table 4.28 0x87 Ticket Validated Field Descriptions

Name	Value	Description
Transaction ID	0 to 255	Transaction ID of the note acceptor event. Each event MUST be acknowledged by the host before it is removed from the stack. Events MUST be protected from power failure.
Index	1 to 255	Packet sequence counter for multi-packet reports.
Size	0 to 60	Number of bytes of Data to follow. The last packet MUST contain less than 60 characters.
Data	0 to 255	Bar code information encoded in ASCII or UTF-16LE. See above.

4.17 Power Failure During Note/Ticket Processing

This section defines the action taken by both the host and note acceptor when power is cycled or the system is reset while a note or ticket transaction is in process. Section 2.4.5 Power Up Sequence, defines the rules of engagement between host and device upon reapplication of power.

In the following examples, whenever *resetting* is mentioned, it is assumed that power is properly applied to the device and that no failures are present.

4.17.1 Note/Ticket Transaction

Correction in v1.3

A **note or ticket transaction** begins when a (0x86 or 0x87) Note or Ticket Validated event is sent by the note acceptor. This is true even if the note acceptor's (0x86 or 0x87) Note or Ticket Validated event has not been fully transmitted by the note acceptor.

The note acceptor MUST consider the transaction ended when the following messages have been ACKed by the host:

- 0x85 Failure Message
- 0x88 Note/Ticket Status
- 0x89 Stacker Status indicating Jam or Fault

Note: The note acceptor must decode and understand the ACK.

The host MUST consider a transaction ended when it receives a:

- 0x85 Failure Message
- 0x88 Note/Ticket Status
- 0x89 Stacker Status indicating Jam or Fault

The note acceptor MUST NOT start a new transaction until the previous transaction has ended.

4.17.2 Note Acceptor Automatic Power up Response

After resetting, the note acceptor will either automatically reject or automatically accept the note or ticket in process.

4.17.2.1 Note Acceptor Automatic Reject

Correction in v1.3

If the note or ticket is accessible after resetting, the note acceptor MUST clear the transport path and report a 0x88 Note/Ticket Status to the host as closure of the transaction. This is true even if the host has instructed the note acceptor to accept the note before the power cycle or system reset.

4.17.2.2 Note Acceptor Automatic Accept

Assuming the note acceptor has been instructed to accept the note or ticket prior to a power cycle or system reset and the note or ticket is no longer accessible, the note acceptor MUST automatically accept the document and report Accepted in the 0x88 Note/Ticket Status report to the host as closure to the transaction.

4.17.3 Power Loss Scenarios

Reference Figure 4.1.

4.17.3.1 Lost Note Acceptor (0x86 or 0x87) Note/Ticket Validated Event or Host ACK Response

1. Note acceptor has sent a (0x86 or 0x87) Note/Ticket Validated event.
2. Power cycle or system reset. The host does not receive the message or the note acceptor does not receive the host ACK.
3. After resetting, the note acceptor will reject the note or ticket.
4. The note acceptor MUST repeat the (0x86 or 0x87) Note/Ticket Validated event with the same TID that was not ACKed. This is required to maintain the TID sequence and ensure the next message sent is not ignored by the host as a resend of a TID.
5. The note acceptor MUST then report a 0x88 Note/Ticket Status indicating Rejected.

4.17.3.2 Lost Host 0x83 Accept Note/Ticket Message

1. Host has received the note acceptors's (0x86 or 0x87) Note/Ticket Validated event.
2. Host ACKs.
3. Host sends a 0x83 Accept Note/Ticket message.

4. Power cycle or system reset. The note acceptor does not receive the 0x83 Accept Note/Ticket message.
5. After resetting, the note acceptor MUST reject the note or ticket and report a 0x88 Note/Ticket Status indicating Rejected.

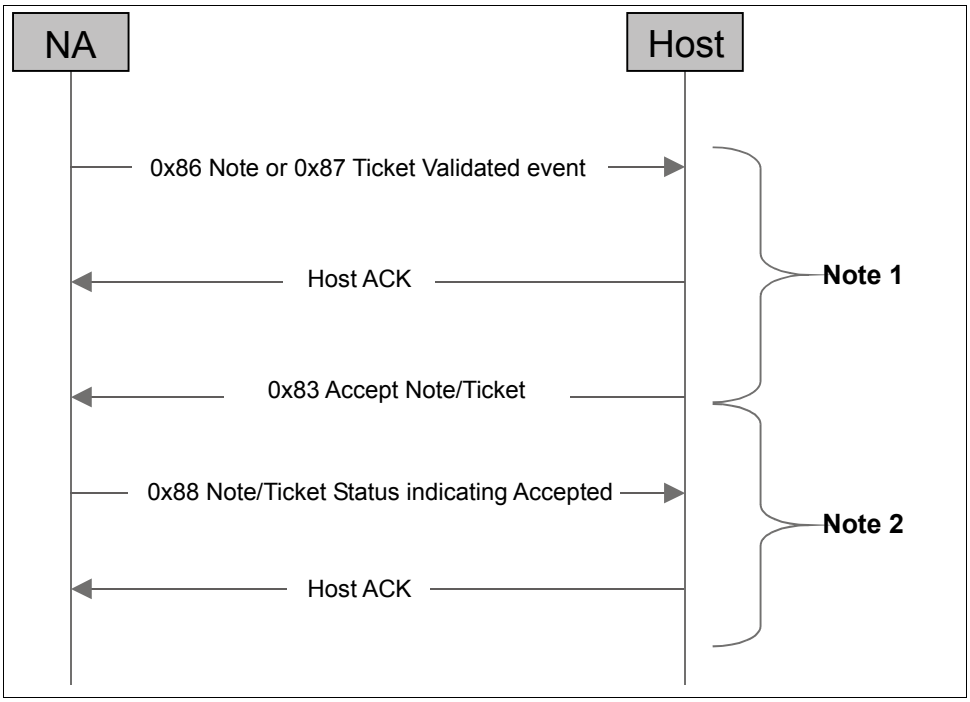
4.17.3.3 Lost Note Acceptor Note/Ticket Status Indicating Accepted or Host ACK Message

1. Note acceptor sends a 0x88 Note/Ticket Status indicating Accepted (and stacked).
2. Power cycle or system reset. The host does not receive the message, or the note acceptor does not receive the host's ACK message.
3. After resetting, the note acceptor will accept the note or ticket and report 0x88 Note/Ticket Status indicating Accepted.

4.17.3.4 Notes

1. The note acceptor is not required to wait for any instructions from the host before accepting or rejecting a note or ticket after resetting. However, the note acceptor MUST communicate the status of the note or ticket.
2. The 0x88 Note/Ticket Status message indicating Accepted does not include the note value or ticket number. Therefore, the host is responsible for saving the note value and ticket number through a power loss or system reset.
3. The note acceptor MUST send a 0x88 Note/Ticket Status indicating Note Path Clear if the note or ticket in process is no longer present after power cycle or system reset. This will ensure the host will not be left in a "Note in Transaction State".

Figure 4.1 Note Acceptor/Host Sequence Diagram



Notes:

1. After resetting, the note acceptor will reject and report Rejected.

2. The note acceptor will either reject and report Rejected or stack and report Accepted.

It is possible that host will receive two (or more) 0x86 Note Validated or 0x87 Ticket Validated events during power failure/reset recovery. The host MUST only issue credit once per Note Validated event or Ticket Validated event. The host MUST ACK and ignore multiple Note Validated events or Ticket Validated events.

Chapter 5

Number Allocation and Report Summary

5.1 Number Allocation and Report Summary

The following sections, Section 5.2 and Section 5.3, identify the usage number and report ID allocation in the note acceptor message protocol, as well as high-level allocation in GDS peripheral device message protocols in general.

Section 5.4 describes the Page 43 and identifies the specific usage number and report ID assignment to each unique type of data in the note acceptor message protocol.

5.2 Usage Number Allocation

All values in hex.

92	GSA Gaming Device
00 to 3F	Top-level device classes
40 to FF	Common reports
0100 to 01FF	Coin Acceptors
0200 to 02FF	Bill Validators
0300 to 03FF	Hoppers
0400 to 04FF	Printers
0500 to 05FF	Touch Screens
0600 to 068F	Card Readers

Allocated Devices

00 – 0F	Unused
11	Coin Acceptor
12	Bill Acceptor
13	Hopper
14	Printer
15	Touch Screen
16	Card Reader
17 – 3F	Reserved

Common report convention…	Bill acceptor convention…
40 to 5FReport usage	0200 to 020FUnused
60 to 8FByte data	0210 to 022FReport usage
90 to AFBit data	0230 to 025FByte data
B0 to CFMiscellaneous data	0260 to 027FBit data
D0 to FFReserved	0280 to 028FMiscellaneous data
	0290 to 02FFReserved

5.3 Report ID Allocation

All Values in Hex

Report ID are in the range of 00 – FF

00	Unused
01 – 4F	Common report IDs
50 – FF	Device specific report IDs

5.4 Report Summary Table

The initial rows of the report summary table (Table 5.1) list the high-level usage numbers identifying all peripheral devices (92) and each of the peripheral device types.

The remaining rows present note acceptor report data organized by report ID, and then by field order within a report. Column descriptions:

Item: Incrementing row identifier, listed only in rows that contain a report ID.

Host Command/Device Events: Report name of command or report. Using two columns provides a visual clue to allow quicker searches.

Report ID: Hexadecimal value of the identifier assigned to the command or event.

Label: Data field name. Used only for data packets other than the command or event report ID.

Usage: Hexadecimal value assigned to a data packet. All unique data packets are assigned a usage number.

NOTE:

The alignment of the usage number (**left**, **middle**, **right**) provides a visual clue as to the category of data:

- left aligned: Report identifier
- middle aligned: one or more bytes.

- right aligned: one or more bits

Size: Size of the data packet in bytes, unless otherwise specified.

Comments: Comments.

Table 5.1 Report Summary (Sheet 1 of 4)

Host Command	Device Event	Report ID (Hex)	Label	Usage (Hex)	Size (Bytes)	Comments
-	-	-	-	92	-	GSA Gaming Device
-	-	-	-	11	-	Coin Acceptor
-	-	-	-	12	-	Bill Validator
-	-	-	-	13	-	Hopper
-	-	-	-	14	-	Printer
-	-	-	-	15	-	Touch Screen
-	-	-	-	16	-	Card Reader
ACK	-	01	-	40	-	
-	-	-	Resync	90	1 bit	
-	-	-	Trans ID	60	1	
Enable	-	02	-	41	-	
Disable	-	03	-	42	-	
Self Test	-	04	-	43	-	
-	-	-	NVM	93	1 bit	
Request GAT Report	-	05	-	44	-	
-	Power Status	06	-	45	-	
-	-	-	Ext. Power	91	1 bit	
-	-	-	Need Reset	92	1 bit	
-	GAT Data	07	-	46	-	
-	-	-	Index	61	1	
-	-	-	Size	62	1	Length of valid data in the data dump.
-	-	-	Data Dump	B0	61	Specific Data
Calculate CRC	-	08	-	47	-	
-	-	-	Seed	63	4	
-	CRC Data	09	-	48	-	
-	-	-	Result	64	4	
-	Device State	0A	-	49	-	
-	-	-	Enable	94	1 bit	

Table 5.1 Report Summary (Sheet 2 of 4)

Host Command	Device Event	Report ID (Hex)	Label	Usage (Hex)	Size (Bytes)	Comments
-	-	-	Disable	95	1 bit	
Number of Note Data Entries	-	80	-	0210	-	
-	Number of Note Data Entries	80	-	0210	-	
-	-	-	Number	0230	1	
Read Note Table	-	81	-	0211	-	
-	Read Note Table	81	-	0211	1	
			Note ID	0237	1	
-	-	-	Currency	0231	3	
-	-	-	Value	0232	2	
-	-	-	Scalar	0260	7 bits	
-	-	-	Sign	0261	1 bit	
-	-	-	Version	0233	1	
Extend Timeout	-	82	-	0212	-	
Accept Note/ Ticket	-	83	-	0213	-	
Return Note/ Ticket	-	84	-	0214	-	
-	Failure Status	85	-	0215	-	
-	-	-	Firmware	0262	1 bit	
-	-	-	Mechanical	0263	1 bit	
-	-	-	Optical	0264	1 bit	
-	-	-	Component	0265	1 bit	
-	-	-	NVM	0266	1 bit	
-	-	-	Unused_0	0267	1 bit	
-	-	-	Unused_1	0268	1 bit	
-	-	-	Other	0269	1 bit	
-	-	-	Diagnostics	0234	1	
-	Note Validated	86	-	0216	-	
-	-	-	Transaction ID	60	1	
-	-	-	Note ID	0235	1	

Table 5.1 Report Summary (Sheet 3 of 4)

Host Command	Device Event	Report ID (Hex)	Label	Usage (Hex)	Size (Bytes)	Comments
-	Ticket Validated	87	-	0217	-	
-	-	-	Transaction ID	60	1	
-	-	-	Length	0236	1	
-	-	-	Code	0280	24	
-	Note/Ticket Status	88	-	0218	-	
-	-	-	Transaction ID	60	1	
-	-	-	Accepted	026A	1bit	
-	-	-	Returned	026B	1bit	
-	-	-	Rejected	026C	1bit	
-	-	-	Removed	0275	1bit	
-	-	-	Note Path Clear	0276	1bit	
-	-	-	Unused_2	026D	1bit	
-	-	-	Cheat	026E	1bit	
-	-	-	Jam	026F	1bit	
-	Stacker Status	89	-	0219	-	
-	-	-	Transaction ID	60	1	
-	-	-	Disconnect	0270	1bit	
-	-	-	Full	0271	1bit	
-	-	-	Jam	0272	1bit	
-	-	-	Unused_0	0267	1bit	
-	-	-	Unused_1	0268	1bit	
-	-	-	Unused_2	026D	1bit	
-	-	-	Unused_3	0273	1bit	
-	-	-	Fault	0274	1bit	
Extension in v1.2						
-	UTF Ticket Validated	8B	-	021B	-	
-	-	-	Transaction ID	60	1	
-	-	-	Index	023B	1	
-	-	-	Size	023C	1	

Table 5.1 Report Summary (Sheet 4 of 4)

Host Command	Device Event	Report ID (Hex)	Label	Usage (Hex)	Size (Bytes)	Comments
-	-	-	Data	023D	60	
Read Note Acceptor Metrics	-	8A	-	021A	-	-
-	Read Note Acceptor Metrics	8A	-	021A	-	-
-	-	-	Index	238	1	-
-	-	-	Size	239	1	-
-	-	-	Data	23A	61	-

Chapter 6

USB Identification Strings

6.1 Overview

Correction in v1.3

USB provides certain information as part of the base protocol (Non-HID devices as well).

idVendor	2 bytes
idProduct	2 bytes
iManufacturer	index of string descriptor
iInterface	index of string descriptor
iSerialNumber	index of string descriptor
iConfiguration	index of string descriptor

UTF-16LE is used for all strings, limiting strings to 126 characters:

```
int((255-2)/2) = 126
```

The two bytes subtracted in the above calculation (-2) are used as follows:

1 byte = string length, in bytes not characters

1 byte = USB DESCRIPTOR constant (0x00)

6.2 idVendor

Unique code supplied to USB member companies. If you do not yet have a code then you need to subscribe to the USB Implementers Forum.

Visit http://www.usb.org.

6.3 idProduct

Each type of product supplied by a manufacturer MUST have a unique ID code (2 bytes).

Windows drivers are loaded on the basis of a unique idVendor (seeSection 6.2), idProduct and iSerialNumber (seeSection 6.6).

6.4 iManufacturer

May be used to give a UTF-16LE representation of the idVendor. This is assigned by each manufacturer and kept consistent with regards to case and spelling.

For example:

```
GSA Member Company Name
```

6.5 iInterface

A string that MUST be returned in this format:

```
<Protocol Level>,<Product Name>,<Firmware Issue>,<Build Version>,<Manufacturing Date>
```

UTF-16LE string made up of several comma-delimited sub-strings. Redundant, trailing commas MAY be omitted. Leading and trailing spaces within sub-strings MUST be ignored.

For example:

```
1.1.1,ProductName,A,1.01,2004-01-01
```

Only the first four items are REQUIRED. At a minimum we could have...

```
1.1.1,ProductName,1A2B3C, 1.01
```

6.5.1 Protocol Level - Major

REQUIRED field, along with the Protocol Level - Minor field (see Section 6.5.2) and Errata Number (see Section 6.5.3).

This field represents the **first** value in the GDS device protocol version, of the format **X.y.z**. This begins at 1 for the initial version of the protocol, and thereafter increments for major revisions of the protocol. A future version of the protocol may include additional commands or events but must be a superset of previous versions allowing full machine compatibility.

6.5.2 Protocol Level - Minor

REQUIRED field, along with the Protocol Level - Major field (see Section 6.5.1) and Errata Number (see Section 6.5.3)

This field represents the **second** value in the GDS device protocol version, of the format **X.y.z**. This begins at 0 for the initial version of the protocol, and thereafter increments for future minor revisions of the protocol. A future version of the protocol may include additional commands or events but must be a superset of previous versions allowing full machine compatibility.

6.5.3 Protocol Level - Errata Number

REQUIRED field, along with the Protocol Level - Major field (see Section 6.5.1) and Protocol Level - Minor field (see Section 6.5.2).

This field represents the **third** value in the GDS device protocol version, of the format **X.y.z**. This begins at 0 for the initial version of the protocol, and thereafter increments for future errata changes of the protocol.

6.5.4 Product Name

REQUIRED field.

Any manufacturer-specific UTF-16LE string, without commas.

This is in effect a UTF-16LE representation of idProduct.

6.5.5 Firmware Issue

REQUIRED field.

Any manufacturer-specific UTF-16LE string, without commas.

6.5.6 Build Version

Correction in v1.3

REQUIRED field.

Any manufacturer-specific UTF-16LE string (without commas).

A standard build could be indicated as STD or 1.01 for example.

6.5.7 Manufacturing Date

OPTIONAL field.

MUST conform to the ISO 8601 standard: ccyy-mm-dd.

The manufacturing date can be defined in a number of different ways. It may or may not relate to product warranty. Some manufacturers find it useful.

6.6 iSerialNumber

REQUIRED field.

Serial numbers MUST be returned as an UTF-16LE string (126 character limit), such as `12345678`.

Leading zeros are acceptable, for example `00000123`.

For every manufacturer, this value MUST be unique for each of the manufacturer's products. This means that two products, each from a different manufacturer, may have the same serial number, but **no** two products from the **same** manufacturer may have the same serial number.

6.7 iConfiguration

Extension in v1.3

REQUIRED field.

A UTF-16LE comma-delimited string that MUST be returned in the following format:

```
<Usage Page>, <Device ID>
```

The values of <Usage Page> and <Device ID> MUST be cast into hexadecimal string representations as shown in the following table:

Table 6.1 iConfiguration Strings

Device Type	iConfiguration String
Coin Acceptor	0x92, 0x11
Bill Acceptor	0x92, 0x12
Hopper	0x92, 0x13
Printer	0x92, 0x14
Touch Screen	0x92, 0x15
Card Reader	0x92, 0x16
Media Window	0x92, 0x17

6.8 Peripheral Class

For HID devices, the reporting structure identifies the peripheral class

```
92    GSA Gaming Device

11    Coin Acceptor
12    Bill Validator
13    Hopper
14    Printer
15    Touch Screen
16    Card Reader
```

The host machine therefore knows what type of equipment it is dealing with and can interrogate the corresponding usage numbers.

6.9 Identifying Firmware Releases

Correction in v1.3

For CRC commands, as well as other software authentication algorithms, to be effective, a host needs to be able to uniquely identify the firmware installed on a device so that it can accurately determine whether the results received from the device match the expected results for the calculation. Thus, it is important that the information provided by a device in the USB identification strings uniquely identify a specific release of the firmware. In particular, the combination of idVendor, idProduct, Firmware Issue, and Build Version MUST be unique.

When used in conjunction with the G2S protocol, these fields are converted into strings and then concatenated together to form a unique identifier for the firmware. The unique identifier uses the following format:

<vendorId>_<productId>_<firmwareIssue>_<buildVersion>

where:

- <vendorId> is the hexadecimal representation of the 2-byte idVendor field.
- <productId> is the hexadecimal representation of the 2-byte idProduct field.
- <firmwareIssue> is the contents of the Firmware Issue field with leading and trailing spaces removed.
- <buildVersion> is the contents of the Build Version field with leading and trailing spaces removed.

The maximum length of the resulting string MUST NOT exceed 255 (two-hundred fifty-five) characters.

For example, the unique identifier for a device with idVendor = 0x1A2B, idProduct = 0x03BF, Firmware Issue = 1A2B3C, and Build Version = 1.01 would be:

1A2B_03BF_1A2B3C_1.01

Chapter 7

Metrics Command Data Format

Extension in v1.2

7.1 Introduction

This chapter details the format of the data included in the 0x8A Read Note Acceptor Metrics report, which the note acceptor sends in response to a 0x8A Read Note Acceptor Metrics report sent by the host. It describes the note acceptor information—that is, note acceptor metrics—that the EGM must know in order to properly understand the note acceptor's capability.

7.2 Metrics Report Data Structure

Note acceptor metrics information is tag-based and MUST use ASCII encoding. All note acceptor metrics data is encased inside the <Metrics></Metrics> start and end tags. At least one instance of each child element is REQUIRED. These are listed below and described in more detail in the following sections:

A complete example of the Metrics element is provided in Section 7.5.

Table 7.1 Read Note Acceptor Metrics Report Data Structure

Child Element Printer Metric Page	Page
RBS Barcode symbologies (standards) support	Page 56
UTF UTF-16 support	Page 57

7.3 RBS: Report Barcode Support

The note acceptor uses the RBS element to indicate the barcode symbologies (standards) the note acceptor supports.

7.3.1 Structure

The following is the structure of the RBS element, where each idvalue is a two-digit ASCII identifier for one barcode symbology (see Section 7.3.2). The values MUST be in ascending numeric order. Leading zeros MUST be included.

```
<RBS>idvalue1 idvalue2 … idvaluen</RBS>
```

7.3.2 Barcode Symbology Indexes

Table of indexes to barcode symbologies:

Table 7.2 Report Barcode Support Indexes (Sheet 1 of 2)

Code	Barcode Symbology	Code	Barcode Symbology
01	Interleaved 2 of 5 (ITF) *Required*	14	EAN-13 + 5
02	Code-128 A *Required*	15	UPC-A
03	Code 39	16	UPC-A + 2
04	Code 39 Check	17	UPC-A + 5

Table 7.2 Report Barcode Support Indexes (Sheet 2 of 2)

Code	Barcode Symbology	Code	Barcode Symbology
05	2 of 5	18	Code-128 B
06	Code 93	19	Code-128 C
07	Codabar	20	EAN-128 A
08	POSTNET	21	EAN-128 B
09	EAN-8	22	EAN-128 C
10	EAN-8 + 2	23	PDF-417 (Text encoding only)
11	EAN-8 + 5		
12	EAN-13		
13	EAN-13 + 2		

7.3.3 Host Behavior

The barcode symbology identifiers MUST be drawn from the values in Table 7.2. If a host system receives an identifier outside of this list, the host MUST discard that value. Additionally, a minimum level of support for barcode symbologies is defined as support for identifiers 01 and 02. If an RBS element is received that does not contain at least these two indicators, the host MUST discard all values in the RBS element and interpret the barcode support as only Interleaved 2 of 5 (identifier 01).

The host MUST also discard any identifiers in the RBS element and interpret the barcode support as only Interleaved 2 of 5 (identifier 01) if:

- The host receives an RBS element where the barcode symbology identifiers contained in the RBS element are not in ascending numeric order.
- The host receives any barcode symbology identifiers that do not consist of two (2), and only two, ASCII digits.
- The host receives an RBS element containing no RBS character set identifiers.
- The host receives a 0x8A Read Note Acceptor Metrics report that does not contain an RBS element.
- The host does not receive a 0x8A Read Note Acceptor Metrics report in response to a 0x8A Read Note Acceptor Metrics command.

7.3.4 Example

```
<RBS> 01 </RBS>
```

7.4 UTF: UTF-16 Support

The note acceptor uses the UTF element to indicate the specific set(s) of UTF characters supported. Each available UTF character set is identified by a 2-digit ASCII character set identifier in the range of 00 – 99. Leading zero (0) characters MUST NOT be dropped or omitted. Supported character set identifiers MUST be

drawn from the sets listed in Table 7.3. All supported character sets MUST be included in a single UTF element.

If a note acceptor includes a character set identifier greater than zero (00) in the UTF element then it MUST correctly read and report all the characters defined in that character set using the UTF-16LE encoding method. Multiple supported character set identifiers MAY be included in a single UTF element.

Minimum UTF support is defined as support for character set identifiers 01, 02, and 09. If a note acceptor exposes UTF support, it MUST support character sets identified by identifiers 01, 02, and 09 and MAY support other character sets using additional character set identifiers from Table 7.3. If a note acceptor includes any character set identifiers that are greater than zero (00), it MUST NOT include character set identifier zero (00). If a note acceptor does not support the characters defined by character set identifiers 01, 02, and 09, it MUST return a single character set identifier zero (00) (No UTF Support) in the UTF element.

The supported character set identifiers MUST be organized in the UTF element in ascending numeric order.

The GDS Note Acceptor and Printer standards are intended to share the same definitions for UTF Character Set Identifiers.

Table 7.3 UTF Character Set Identifiers

UTF Character Set Identifier	Unicode Script	Unicode Range
00	No Support	None
01	Basic Latin	0000-007F
02	Latin – 1	0080-00FF
03	Latin Extended A	0100 – 017F
04	Latin Extended B	0180 – 024F
09	Currency Symbols	20A0 – 20B5
10	Bopomofo (Mandarin based)	3100 – 312F
11	Bopomofo Extended	31A0 – 31BF
20	Cyrillic	0400 - 04FF
21	Cyrillic Supplement	0500 – 052F
25	Hiragana (Japanese)	3040 – 309F
25	Katakana (Japanese)	30A0 – 30FF
30	Hebrew	0590 – 05FF
95	Runic	16A0 – 16FF
99	Full UTF Support (all UTF Identifiers supported)	

7.4.1 Host Behavior

UTF character set identifiers MUST be drawn from the values in Table 7.3. If a host system receives an identifier outside of this list, the host MUST discard that value. Additionally, a minimum level of support for UTF characters is defined as support for character set identifiers 01, 02, and 09. If a UTF element is received that does not contain at least these three indicators, the host MUST discard all values in the UTF element and interpret the response as No UTF Support (identifier 00).

The host MUST also discard any character set identifiers in the UTF element and interpret the UTF element as containing a single character set identifier zero (00) (No UTF Support) if:

- The host receives a UTF element containing the character identifier 00 in any position in the UTF element regardless of any other character set identifiers present in the UTF element.
- The host receives a UTF element where the character set identifiers contained in the UTF element are not in ascending numeric order.
- The host receives any character set identifiers that do not consist of two (2), and only two, ASCII digits.
- The host receives a UTF element containing no UTF character set identifiers.
- The host receives a 0x8A Read Note Acceptor Metrics report that does not contain a UTF element.
- The host does not receive a 0x8A Read Note Acceptor Metrics report in response to a 0x8A Read Note Acceptor Metrics command.

7.4.2 Structure

The following is the structure of the UTF element. The utfId is a 2-digit ASCII number in the range of 00 – 99 that corresponds to a specific UTF character set identifier listed in Table 7.3. Leading zeros in the utfId MUST NOT be dropped or omitted. If multiple character sets are supported they MUST be included in a single UTF element. The character set identifiers MUST be in ascending numeric order in the UTF element.

```
<UTF> utfId1, utfId2... utfIdn </UTF>
```

7.4.3 Examples

The following examples demonstrate how UTF elements are constructed.

```
<UTF> 01 02 09 </UTF> //Minimum UTF support
<UTF> 00 </UTF> //No UTF Support
<UTF>01 02 03 09</UTF> //Minimum UTF support plus Latin Extended A
```

7.4.4 Example that maps C0 Controls and Basic Latin (0000-007F) and all Indic Scripts (0900-0DFF)

```
<UTF> 000000 00007F 000900 000DFF </UTF>
```

7.5 Example: Data Structure

The following is an example of the structure of the note acceptor metrics data included in the 0x8A report.

```
<Metrics>
   // Barcode support. ITF is minimum required support.
   <RBS> 01 </RBS>
   // Note acceptor UTF-16 support
   <UTF> 01 02 09 </UTF>
```

```
</Metrics>
```

Glossary

Correction in v1.3

Diagnostic Commands The following reports are diagnostic commands:

Report 0x04 Self Test

Report 0x05 Request GAT Report

Report 0x08 Calculate CRC

Report 0x80 Number of Note Data Entries

Report 0x81 Read Note Table

Disabled Condition of a peripheral device imposed by its host in which the device does not perform operations other than diagnostic functions and communications in response to host requests. A device that is disabled by the host must be re-enabled by the host. See also *Self-disabled*.

GAT Game Authentication Terminal. An application, installed on a personal computer or laptop, which uses serial communication to authenticate game program components on a gaming machine.

Host A machine that gathers data from, and sends directives to, its peripheral devices.

NVM Non-volatile memory.

Report ID A unique identifier assigned by GSA to GDS device commands and events.

Self-Disabled Identical to *Disabled*, except the peripheral device imposes the condition on itself in response to detecting a fault within itself; and if the fault is corrected, the device must remain disabled until enabled by the host.

TID event Any event that contains a Transaction ID number in its report format. A pending TID event is a TID event that is sent to the host but has not yet been acknowledged by the host. See also *Transaction ID*.

Token A chip resembling a coin to which a casino may assign a certain monetary value or number of credits.

Transaction ID A unique identifier assigned to a communication transaction between a host and a peripheral device. See also *TID event*.

True Coin A coin of the proper denomination, origin, material, and mold that is intended to be accepted by the device.

Path, Intended/Verified Intended Path: set by the device's host for an inserted coin to go through.

Verified Path: detected by sensory equipment that an inserted coin had taken.

Appendix A

CRC-32 Checksum Calculation

A.1 CRC-32 Checksum Calculation

Correction in v1.3

Procedure for CRC calculations:

1. The method shall conform to Autodin 2, also known as CRC-32;
2. The polynomial shall be $X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$;
3. The calculation shall be performed least significant bit first;
4. To allow the continuation of multiple CRC32 calculation over multiple image files the final retuned remainder or result must NOT be complemented.
5. The following 62 byte string (without quotes "") can be used to verify the CRC32 implementation with the seeds and results.

   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"

   Which can also be represented at the follow C array:

```
const unsigned short testData[62] = {
0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f, 0x50,
0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
0x59, 0x5a, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66,
0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e,
0x6f, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76,
0x77, 0x78, 0x79, 0x7a, 0x30, 0x31, 0x32, 0x33,
0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
```

   Seed of **0x00000000** results in **0x02A641B3**

   Seed of **0x00001234** results in **0xE802F72A**

   Seed of **0x0000ABCD** results in **0x09DEEB32**

   Seed of **0xFFFFFFFF** results in **0xE03D192D**

A.2 C Language Example

Correction in v1.3

```
/*************************************************************/
/* C LANGUAGE VERSIONS */
/*************************************************************/
/*DISCLAIMER: */
/*This code is offered for example only. It is the user's */
/* responsibility to verify correct operation in a particular */
/* application. */
/* */
/* Compiler optimization is getting better but probably */
/* won't do as well as hand-coded ASM. Some careful */
/* examination of the compiled code may be required to */
/* get acceptable performance, particularly with the */
/* shift register method. */
/* */
/*************************************************************/
/*****************************************/
/* */
/* Example CRC algorithm with lookup table */
/* */
/*****************************************/
#include <stdio.h>
#include <stdlib.h>

#define _usage                                          \
 "\nUsage: %s [FILE] [OPTION...SEED]\n\n"\
 "SEED of 0x00 if no SEED provided\n\n"

const unsigned long crc_table[] = {
0x00000000,0x77073096,0xee0e612c,0x990951ba,0x076dc419,0x706af48f,0xe963a535,0x9e6495a3,
0x0edb8832,0x79dcb8a4,0xe0d5e91e,0x97d2d988,0x09b64c2b,0x7eb17cbd,0xe7b82d07,0x90bf1d91,
0x1db71064,0x6ab020f2,0xf3b97148,0x84be41de,0x1adad47d,0x6ddde4eb,0xf4d4b551,0x83d385c7,
0x136c9856,0x646ba8c0,0xfd62f97a,0x8a65c9ec,0x14015c4f,0x63066cd9,0xfa0f3d63,0x8d080df5,
0x3b6e20c8,0x4c69105e,0xd56041e4,0xa2677172,0x3c03e4d1,0x4b04d447,0xd20d85fd,0xa50ab56b,
0x35b5a8fa,0x42b2986c,0xdbbbc9d6,0xacbcf940,0x32d86ce3,0x45df5c75,0xdcd60dcf,0xabd13d59,
0x26d930ac,0x51de003a,0xc8d75180,0xbfd06116,0x21b4f4b5,0x56b3c423,0xcfba9599,0xb8bda50f,
0x2802b89e,0x5f058808,0xc60cd9b2,0xb10be924,0x2f6f7c87,0x58684c11,0xc1611dab,0xb6662d3d,
0x76dc4190,0x01db7106,0x98d220bc,0xefd5102a,0x71b18589,0x06b6b51f,0x9fbfe4a5,0xe8b8d433,
0x7807c9a2,0x0f00f934,0x9609a88e,0xe10e9818,0x7f6a0dbb,0x086d3d2d,0x91646c97,0xe6635c01,
0x6b6b51f4,0x1c6c6162,0x856530d8,0xf262004e,0x6c0695ed,0x1b01a57b,0x8208f4c1,0xf50fc457,
0x65b0d9c6,0x12b7e950,0x8bbeb8ea,0xfcb9887c,0x62dd1ddf,0x15da2d49,0x8cd37cf3,0xfbd44c65,
0x4db26158,0x3ab551ce,0xa3bc0074,0xd4bb30e2,0x4adfa541,0x3dd895d7,0xa4d1c46d,0xd3d6f4fb,
0x4369e96a,0x346ed9fc,0xad678846,0xda60b8d0,0x44042d73,0x33031de5,0xaa0a4c5f,0xdd0d7cc9,
0x5005713c,0x270241aa,0xbe0b1010,0xc90c2086,0x5768b525,0x206f85b3,0xb966d409,0xce61e49f,
0x5edef90e,0x29d9c998,0xb0d09822,0xc7d7a8b4,0x59b33d17,0x2eb40d81,0xb7bd5c3b,0xc0ba6cad,
0xedb88320,0x9abfb3b6,0x03b6e20c,0x74b1d29a,0xead54739,0x9dd277af,0x04db2615,0x73dc1683,
0xe3630b12,0x94643b84,0x0d6d6a3e,0x7a6a5aa8,0xe40ecf0b,0x9309ff9d,0x0a00ae27,0x7d079eb1,
0xf00f9344,0x8708a3d2,0x1e01f268,0x6906c2fe,0xf762575d,0x806567cb,0x196c3671,0x6e6b06e7,
0xfed41b76,0x89d32be0,0x10da7a5a,0x67dd4acc,0xf9b9df6f,0x8ebeeff9,0x17b7be43,0x60b08ed5,
0xd6d6a3e8,0xa1d1937e,0x38d8c2c4,0x4fdff252,0xd1bb67f1,0xa6bc5767,0x3fb506dd,0x48b2364b,
0xd80d2bda,0xaf0a1b4c,0x36034af6,0x41047a60,0xdf60efc3,0xa867df55,0x316e8eef,0x4669be79,
0xcb61b38c,0xbc66831a,0x256fd2a0,0x5268e236,0xcc0c7795,0xbb0b4703,0x220216b9,0x5505262f,
0xc5ba3bbe,0xb2bd0b28,0x2bb45a92,0x5cb36a04,0xc2d7ffa7,0xb5d0cf31,0x2cd99e8b,0x5bdeae1d,
0x9b64c2b0,0xec63f226,0x756aa39c,0x026d930a,0x9c0906a9,0xeb0e363f,0x72076785,0x05005713,
0x95bf4a82,0xe2b87a14,0x7bb12bae,0x0cb61b38,0x92d28e9b,0xe5d5be0d,0x7cdcefb7,0x0bdbdf21,
0x86d3d2d4,0xf1d4e242,0x68ddb3f8,0x1fda836e,0x81be16cd,0xf6b9265b,0x6fb077e1,0x18b74777,
0x88085ae6,0xff0f6a70,0x66063bca,0x11010b5c,0x8f659eff,0xf862ae69,0x616bffd3,0x166ccf45,
0xa00ae278,0xd70dd2ee,0x4e048354,0x3903b3c2,0xa7672661,0xd06016f7,0x4969474d,0x3e6e77db,
0xaed16a4a,0xd9d65adc,0x40df0b66,0x37d83bf0,0xa9bcae53,0xdebb9ec5,0x47b2cf7f,0x30b5ffe9,
0xbdbdf21c,0xcabac28a,0x53b39330,0x24b4a3a6,0xbad03605,0xcdd70693,0x54de5729,0x23d967bf,
0xb3667a2e,0xc4614ab8,0x5d681b02,0x2a6f2b94,0xb40bbe37,0xc30c8ea1,0x5a05df1b,0x2d02ef8d
};

int main(int argc, char *argv[])
{
    FILE *fp;
    unsigned long crc=0;
    unsigned long  buffer;
    int size=0;
    char *seed;
    char *filename;
```

```
    if (argc < 2) {/* no args; */
           printf(_usage,*argv);
           return 1;
    }else if ((fp = fopen(*++argv, "rb")) == NULL) {
           printf ("\nFailed to open %s\n", *argv);
           return 1;
    }else {
           filename = *argv;
           if (argc>2){
                  seed = *++argv;
                  crc = strtoul(seed, &seed, 16);
           }else{
                  crc = atol(seed);
           }
    printf("\nCRC SEED - 0x%08lX \n", crc);
    while (( buffer = fgetc(fp)) !=(unsigned int)EOF) {
           crc = (crc >> 8) ^ crc_table[buffer ^ (crc & 0xFF)];
           ++size;
           }
    printf("\n%s - %i bytes \n\nCRC32 = 0x%08lX  Complement = 0x%08lX\n\n", filename,size,
crc, ~crc);
    fclose(fp);
    }
    return 0;
}
```

Appendix B

Note Acceptor Report Descriptors

B.1 Note Acceptor Report Descriptors

Correction in v1.3

The following verbose report descriptors have been created to show an implementation of each device class interface as specified in this document.

While these descriptors can be used 'as is' in a device, it should be noted they can be optimized to reduce the storage. An understanding of the HID report structure is required to undertake this activity.

The following report descriptors can be compiled and used by any 8051 compatible micro-controller. For non-compatible 8051 micro-controllers, extract the data bytes and encode as appropriate.

These descriptors were created with the USB-IF HID Report Descriptor tool.

```
db 005h,092h                ; USAGE_PAGE (GSA Gaming Device)
db 009h,012h                ; USAGE (Note/Ticket Acceptor Device)
db 0a1h,001h                ; COLLECTION (Application)

; ACK
db 009h,040h                ;   USAGE - ACK (0x40)
db 085h,001h                ;   REPORT_ID (0x01)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,090h                ;     USAGE - Resync (0x90)
db 0b1h,002h                ;     FEATURE (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,007h                ;     REPORT_COUNT (7)
db 0b1h,003h                ;     FEATURE (Cnst,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,060h                ;     USAGE - Transaction ID (0x60)
db 0b1h,002h                ;     FEATURE (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; Enable
db 009h,041h                ;   USAGE - Enable (0x41)
db 085h,002h                ;   REPORT_ID (0x02)
db 075h,008h                ;   REPORT_SIZE (8)
db 095h,001h                ;   REPORT_COUNT (1)
db 0b1h,003h                ;   FEATURE (Cnst,Var,Abs)

; Disable
db 009h,042h                ;   USAGE - Disable (0x42)
db 085h,003h                ;   REPORT_ID (0x03)
db 075h,008h                ;   REPORT_SIZE (8)
db 095h,001h                ;   REPORT_COUNT (1)
db 0b1h,003h                ;   FEATURE (Cnst,Var,Abs)

; Self Test
db 009h,043h                ;   USAGE - Self-Test (0x43)
db 085h,004h                ;   REPORT_ID (0x04)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;   LOGICAL_MINIMUM (0)
db 025h,001h                ;   LOGICAL_MAXIMUM (1)
db 075h,001h                ;   REPORT_SIZE (1)
db 095h,001h                ;   REPORT_COUNT (1)
db 009h,093h                ;   USAGE - NVM (0x93)
db 0b1h,002h                ;   FEATURE (Data,Var,Abs)
db 015h,000h                ;   LOGICAL_MINIMUM (0)
db 025h,001h                ;   LOGICAL_MAXIMUM (1)
db 075h,001h                ;   REPORT_SIZE (1)
db 095h,007h                ;   REPORT_COUNT (7)
db 0b1h,003h                ;   FEATURE (Cnst,Var,Abs)
db 0c0h                     ;   END_COLLECTION
```

```
; Request GAT Report
db 009h,044h                ;   USAGE - Self-Test (0x44)
db 085h,005h                ;   REPORT_ID (0x05)
db 075h,008h                ;   REPORT_SIZE (8)
db 095h,001h                ;   REPORT_COUNT (1)
db 0b1h,003h                ;   FEATURE (Cnst,Var,Abs)

; Power Status
db 009h,045h                ;   USAGE - Power Status (0x45)
db 085h,006h                ;   REPORT_ID (0x06)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,091h                ;     USAGE - External Power (0x91)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,092h                ;     USAGE - Need Reset (0x92)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,006h                ;     REPORT_COUNT (6)
db 081h,003h                ;     INPUT (Cnst,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; GAT Data
db 009h,046h                ;   USAGE - GAT Data (0x46)
db 085h,007h                ;   REPORT_ID (0x07)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,001h                ;     LOGICAL_MINIMUM (1)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,061h                ;     USAGE - GAT Index number (0x61)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,03dh                                ;     LOGICAL_MAXIMUM (61)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,062h                ;     USAGE - GAT content size (0x62)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,00ah                ;     LOGICAL_MINIMUM (10)
db 026h,07eh,000h           ;     LOGICAL_MAXIMUM (126)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,03dh                ;     REPORT_COUNT(61) - Report length is product
                            ;     specific, indicated in content length
db 009h,0b0h                ;     USAGE - Data dump (0xB0)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; Calculate CRC
db 009h,047h                ;   USAGE - Calculate CRC (0x47)
db 085h,008h                ;   REPORT_ID (0x08)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,004h                ;     REPORT_COUNT (4)
db 009h,063h                ;     USAGE - Seed (0x63)
db 0b1h,002h                ;     FEATURE (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; CRC Data
db 009h,048h                ;   USAGE - CRC Data (0x48)
db 085h,009h                ;   REPORT_ID (0x09)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,004h                ;     REPORT_COUNT (4)
```

```
db 009h,064h                ;     USAGE - Result (0x64)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; Device State
db 009h,049h                ; USAGE - Device State(0x49)
db 085h,00Ah                ; REPORT_ID (0x0A)
db 0a1h,002h                ; COLLECTION (Logical)
db 015h,000h                ; LOGICAL_MINIMUM (0)
db 025h,001h                ; LOGICAL_MAXIMUM (1)
db 075h,001h                ; REPORT_SIZE (1)
db 095h,001h                ; REPORT_COUNT (1)
db 009h,094h                ; USAGE - Enable (0x94)
db 081h,002h                ; INPUT (Data,Var,Abs)
db 015h,000h                ; LOGICAL_MINIMUM (0)
db 025h,001h                ; LOGICAL_MAXIMUM (1)
db 075h,001h                ; REPORT_SIZE (1)
db 095h,001h                ; REPORT_COUNT (1)
db 009h,095h                ; USAGE - Disable (0x95)
db 081h,002h                ; INPUT (Data,Var,Abs)
db 015h,000h                ; LOGICAL_MINIMUM (0)
db 025h,001h                ; LOGICAL_MAXIMUM (1)
db 075h,001h                ; REPORT_SIZE (1)
db 095h,006h                ; REPORT_COUNT (6)
db 081h,003h                ; INPUT (Cnst,Var,Abs)
db 0c0h                     ; END_COLLECTION

; Number of Note Data Entries Command
db 00ah,010h,002h           ;  USAGE - Number of Note Data Entries(0x0210)
db 085h,080h                ; REPORT_ID (0x80)
db 075h,008h                ; REPORT_SIZE (8)
db 095h,001h                ; REPORT_COUNT (1)
db 0b1h,003h                ; FEATURE (Cnst,Var,Abs)

; Number of Note Data Entries Response
db 00ah,010h,002h           ;   USAGE - Number of Note Data Entries(0x0210)
db 085h,080h                ;  REPORT_ID (0x80)
db 0a1h,002h                ;  COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,030h,002h           ;     USAGE - Number(0x0230)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; Read Note Table Command
db 00ah,011h,002h           ; USAGE - Read Note Table(0x0211)
db 085h,081h                ; REPORT_ID (0x81)
db 075h,008h                ; REPORT_SIZE (8)
db 095h,001h                ; REPORT_COUNT (1)
db 0b1h,003h                ; FEATURE (Cnst,Var,Abs)

; Read Note Table Response
db 00ah,011h,002h           ;   USAGE - Read Note Table(0x0211)
db 085h,081h                ;   REPORT_ID (0x81)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,001h                ;     LOGICAL_MINIMUM (1)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,037h,002h           ;     USAGE - Note ID(0x0237)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,041h                ;     LOGICAL_MINIMUM (65['A'])
db 026h, 05ah, 000h         ;     LOGICAL_MAXIMUM (90['z'])
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,003h                ;     REPORT_COUNT (3)
db 00ah,031h,002h           ;     USAGE - Currency(0x0231)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 027h,0ffh,0ffh,000h,000h ;     LOGICAL_MAXIMUM (65535)
db 075h,010h                ;     REPORT_SIZE (16)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,032h,002h           ;     USAGE - Value (0x0232)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
```

```
db 025h,07fh                ;     LOGICAL_MAXIMUM (127)
db 075h,007h                ;     REPORT_SIZE (7)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,060h,002h           ;     USAGE - Scalar(0x0260)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,061h,002h           ;     USAGE - Sign +/-(0x0261)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h, 0ffh, 000h         ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,033h,002h           ;     USAGE - Version(0x0233)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; Extend TimeOut
db 00ah,012h,002h           ;   USAGE – Extend TimeOut(0x0212)
db 085h,082h                ;   REPORT_ID (0x82)
db 075h,008h                ;   REPORT_SIZE (8)
db 095h,001h                ;   REPORT_COUNT (1)
db 0b1h,003h                ;   FEATURE (Cnst,Var,Abs)

; Accept Note/Ticket
db 00ah,013h,002h           ;   USAGE – Accept Note/Ticket(0x0213)
db 085h,083h                ;   REPORT_ID (0x83)
db 075h,008h                ;   REPORT_SIZE (8)
db 095h,001h                ;   REPORT_COUNT (1)
db 0b1h,003h                ;   FEATURE (Cnst,Var,Abs)

; Return Note/Ticket
db 00ah,014h,002h           ;   USAGE – Return Note/Ticket(0x0214)
db 085h,084h                ;   REPORT_ID (0x84)
db 075h,008h                ;   REPORT_SIZE (8)
db 095h,001h                ;   REPORT_COUNT (1)
db 0b1h,003h                ;   FEATURE (Cnst,Var,Abs)

; Failure Status
db 00ah,015h,002h           ;   USAGE - Failure Status (0x0215)
db 085h,085h                ;   REPORT_ID (0x85)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,062h,002h           ;     USAGE - Firmware (0x0262)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,063h,002h           ;     USAGE - Mechanical(0x0263)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,064h,002h           ;     USAGE - Optical (0x0264)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,065h,002h           ;     USAGE - Component(0x0265)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,066h,002h           ;     USAGE – NVM(0x0266)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
```

```
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,067h,002h           ;     USAGE – Unused_0 (0x0267)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,068h,002h           ;     USAGE – Unused_1(0x0268)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,069h,002h           ;     USAGE - Other (0x0269)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,034h,002h           ;     USAGE - Diagnostics (0x0234)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

;Note Validated
db 00ah,016h,002h           ;   USAGE - Note Validated Event(0x216)
db 085h,086h                ;   REPORT_ID (0x86)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,060h                ;     USAGE - Transaction ID(0x60)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h, 001h               ;     LOGICAL_MINIMUM (1)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,035h,002h           ;     USAGE - Note ID(0x0235)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

;Ticket Validated
db 00ah,017h,002h           ;   USAGE - Ticketed Validated Event(0x0217)
db 085h,087h                ;   REPORT_ID (0x87)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,060h                ;     USAGE - Transaction ID(0x60)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,012h                ;     LOGICAL_MINIMUM (18)
db 026h,018h, 000h          ;     LOGICAL_MAXIMUM (24)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,036h,002h           ;     USAGE - Length(0x0236)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh, 000h          ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,018h                ;     REPORT_COUNT (24)
db 00ah,080h,002h           ;     USAGE - Code(0x0280)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

;Note/Ticket Status
db 00ah,018h,002h           ;   USAGE - Note/Ticket Status Event(0x0218)
db 085h,088h                ;   REPORT_ID (0x88)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h, 060h               ;     USAGE - Transaction ID(0x60)
db 081h,002h                ;     INPUT (Data,Var,Abs)
```

```
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,06ah,002h           ;     USAGE - Accepted(0x026A)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,06bh,002h           ;     USAGE - Returned(0x026B)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,06ch,002h           ;     USAGE - Rejected(0x026C)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,075h,002h           ;     USAGE - Removed(0x0275)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,076h,002h           ;     USAGE - Note Path Clear(0x0276)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,06dh,002h           ;     USAGE - Unused_0(0x026D)
db 081h,003h                ;     INPUT (Cnst,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,06eh,002h           ;     USAGE- Cheat(0x026E)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,06fh,002h           ;     USAGE - Jam (0x026F)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

;Stacker Status
db 00ah,019h,002h           ;   USAGE - Stacker Status Event(0x0219)
db 085h,089h                ;   REPORT_ID (0x89)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h, 060h               ;     USAGE - Transaction ID(0x60)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,070h,002h           ;     USAGE Disconnected(0x270)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,071h,002h           ;     USAGE - Full(0x271)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
```

```
db 00ah,072h,002h           ;     USAGE - Jam(0x0272)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,067h,002h           ;     USAGE - Unused_0(0x0267)
db 081h,003h                ;     INPUT (Cnst,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,068h,002h           ;     USAGE - Jam_Unused_1(0x0268)
db 081h,003h                ;     INPUT (Cnst,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,06dh,002h           ;     USAGE - Unused_2(0x026D)
db 081h,003h                ;     INPUT (Cnst,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,073h,002h           ;     USAGE - Unused_3(0x0273)
db 081h,003h                ;     INPUT (Cnst,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,001h                ;     LOGICAL_MAXIMUM (1)
db 075h,001h                ;     REPORT_SIZE (1)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,074h,002h           ;     USAGE - Fault(0x0274)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

;UTF Ticket Validated
db 00ah,01bh,002h           ;   USAGE - Ticketed Validated Event(0x021B)
db 085h,08bh                ;   REPORT_ID (0x8B)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 026h,0ffh,000h           ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 009h,060h                ;     USAGE - Transaction ID(0x60)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,001h                ;     LOGICAL_MINIMUM (1)
db 026h,0ffh, 000h          ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,03bh, 002h          ;     USAGE – Index Number (0x023B)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,03ch                ;     LOGICAL_MAXIMUM (60)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,03ch, 002h          ;     USAGE – Content Size (0x023C)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,0ffh,               ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,03ch                ;     REPORT_COUNT(60)– Max Length of Data
db 00ah,03dh, 002h          ;     USAGE – Bar Code Data (0x023D)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

; Read Note Acceptor Metrics Command
db 00ah, 01ah, 002h         ;   USAGE – Read Note Acceptor Metrics (0x021A)
db 085h, 08ah               ;   REPORT_ID (0x8A)
db 075h, 008h               ;   REPORT_SIZE (8)
db 095h, 001h               ;   REPORT_COUNT (1)
db 0b1h, 003h               ;   FEATURE (Cnst, Var, Abs)

; Read Note Acceptor Metrics Response
db 00ah,01ah, 002h          ;   USAGE – Read Note Acceptor Metrics (0x021A)
db 085h,08ah                ;   REPORT_ID (0x8A)
db 0a1h,002h                ;   COLLECTION (Logical)
db 015h,001h                ;     LOGICAL_MINIMUM (1)
```

```
db 026h,0ffh, 000h          ;     LOGICAL_MAXIMUM (255)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,038h, 002h          ;     USAGE – Read Note Acceptor Metrics Index number (0x0238)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,000h                ;     LOGICAL_MINIMUM (0)
db 025h,03dh                ;     LOGICAL_MAXIMUM (61)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,001h                ;     REPORT_COUNT (1)
db 00ah,039h, 002h          ;     USAGE – Read Note Acceptor Metrics content size (0x0239)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 015h,020h                ;     LOGICAL_MINIMUM (32)
db 025h,07eh,               ;     LOGICAL_MAXIMUM (126)
db 075h,008h                ;     REPORT_SIZE (8)
db 095h,03dh                ;     REPORT_COUNT(61) – Max length of report data
db 00ah,03ah, 002h          ;     USAGE – Read Note Acceptor Metrics Data (0x023A)
db 081h,002h                ;     INPUT (Data,Var,Abs)
db 0c0h                     ;   END_COLLECTION

db 0c0h                    ;   END_COLLECTION
```

Appendix C

Member Comments

C.1 GDSCR-25: Host-Controlled Reverse Reads

CREATED: 23/Aug/13

REPORTER: Ethan Tower

The attached flowcharts illustrate how a card reader can be programmed so the host can control whether a card is read upon removal.

The critical information is in section 2. If the host clears the buffer before the card is removed, the card will be read (the buffer will be updated) upon removal. If the host does not clear the buffer before the card is removed, the card will not be read (the buffer will be cleared).

This allows the host to control whether a read is performed upon removal. If the host clears the buffer then a read is performed. If the host does not clear the buffer then a read is not performed.

Based on the proposed clarifications, is this behavior acceptable? Should this behavior be allowed? Is a clarification needed?

C.1.1 Resolution: Fixed

The committee reviewed the flowchart and decided to not add a requirement that a clear buffer command be sent to the device after a forward read to activate a reverse read. A card reader with reverse read capabilities should read upon insert and removal without host intervention as specified. Further clarifications were added to the protocol indicating that implementations must not overload GDS commands with other features, such as requiring that a clear buffer command be sent to activate reverse reads.

C.2 GDSCR-32: Self Test request/response procedure ambiguities

CREATED: 25/Sept/14

REPORTER: Frederic Fabbri

The following issue applies to all GDS protocols not just the card reader protocol. Self-test request/response procedure is mentioned in 3 sections:

a) section 2.5.5 USB Enumeration Sequence:

a.1)

After the device receives its first Disable command from the host, it responds with the 0x0A Device State event (see Page 26), [and then it reports any event messages in the following priority]

1. Power Status events, if external power is missing.

a.2)

2. [Perform self-test and report failure status] before generating any other reports. While performing selftest a device may NAK, at the USB level, any host commands.

---

b) section 3.5 Report 0x04 Self Test

The device MUST initiate a self-test sequence when instructed by the host. [When the host requests a self-test, the device MUST respond with a 0x62 Failure Status event].

---

c) section 4.8 Report 0x62 Failure Status

c.1)

When instructed by the host, the device MUST perform a diagnostic test. [If any fault condition is detected, this event is sent to the host unless otherwise specified.]

c.2)

4.8.1 Failure Status Rules:

[A 0x62 Failure Status report MUST be sent to the host after receiving a 0x04 Self Test command] (see Page 14), after receiving a 0x02 Enable command (see Page 13), if a fault exists, and as soon as a new fault condition is detected.

I am confused by the sentenced bracketed in [C.1]. This seems to indicate that the device is not required to report of failure status if there are no faults.

However this contradicts [b] and [c.2] which requires the device to always report a failure status, with or without faults present.

Section [a.1] is also ambiguous because the word 'any' seems to imply that power status and failure status may not be reported. While this is true for power status (i.e. bus powered device) , [a.2] however, requires the device to always report a failure status.

I would like to clarify the protocol as follow:

-Remove the word "any" from the sentence "it reports any event messages in the following priority" in section 2.5.5.

- delete the sentence "If any fault condition is detected, this event is sent to the host unless otherwise specified." from section 4.8

C.2.1 Resolution: Fixed

The committee decided, in addition to the changes mentioned above, to change text in the Event Codes section of each protocol to read:

When instructed by the host, the device MUST perform a self-test and then send a Failure Status report to the host reporting the current status of the device.

During normal operation, the device MUST also continually check components that may lead to faulty operation. When doing so, if a fault condition is detected, a Failure Status report MUST be sent to the host unless specified differently elsewhere.

END OF DOCUMENT

Released: TBD