Request #: HUTRR84
Title: Lighting And Illumination
Spec Release: 1.12
Received: 14 May 2018
Requester: Nathan Sherman
Company: Microsoft
Phone:
Email: nathans@microsoft.com

Current Status: Approved 4-0-0
Submitted: 15 May 2018
Voting Ended: 6 June 2018

Lighting and Illumination Page (0x59)

1 Summary

This request is to establish a new Usage Page for Lighting/Illumination devices (in particular LampArray). LampArray devices have one or more Lamps (i.e. lights/LEDs/bulbs, etc…) that can be directly manipulated; setting state (on/off), brightness and color (RGB). While today HID does have an LED page (0x08), it restricts Lamps to specific purposes (e.g. Caps-Lock, Num-Lock) and to specific colors (Red/Green/Amber). The proposed LampArray permits a much richer property model for both the LampArray and individual Lamps and finer control over the Lamp state.

Below is a brief list of observed device classes (in the PC space) that often contain controllable Lamps, using vendor specific protocols (often built by vendors on top of HID)

- Gaming Keyboards
- Gaming Keypads
- Gaming Mice
- Gaming Mousepads
- Headsets
- RAM Sticks
- Motherboards
- CPU Fans
- PC Case Lights
- Light Bulbs

2 Proposal

Add the following to Table 1: Usage Page Summary:

Field	Value
Page ID	0x59
Page Name	LightingAndIllumination
Section or Document	(next available per technical editor)

Edit the appropriate Reserved range in Table 1 in which the Page 0x59 lies to not include Page 0x59.

Create a new section for the Lighting and Illumination Page, with usage table and add the follow to it.

Table (1): Lighting and Illumination Page

Usage ID	Usage Name	Usage Type	Section
0x00	Undefined		
0x01	**LampArray**	**CA**	3.5.1
0x02	**LampArrayAttributesReport**	**CL**	3.5.2
0x03	LampCount	SV/DV	3.5.2
0x04	BoundingBoxWidthInMicrometers	SV	3.5.2
0x05	BoundingBoxHeightInMicrometers	SV	3.5.2
0x06	BoundingBoxDepthInMicrometers	SV	3.5.2
0x07	LampArrayKind	SV	3.5.2
0x08	MinUpdateIntervalInMicroseconds	SV	3.5.2
0x09-1F	Reserved		
0x20	**LampAttributesRequestReport**	**CL**	3.5.3
0x21	LampId	SV/DV	3.5.3
0x22	**LampAttributesResponseReport**	**CL**	3.5.3
0x23	PositionXInMicrometers	DV	3.5.3
0x24	PositionYInMicrometers	DV	3.5.3
0x25	PositionZInMicrometers	DV	3.5.3
0x26	LampPurposes	DV	3.5.3
0x27	UpdateLatencyInMicroseconds	DV	3.5.3
0x28	RedLevelCount	DV	3.5.3
0x29	GreenLevelCount	DV	3.5.3
0x2A	BlueLevelCount	DV	3.5.3
0x2B	IntensityLevelCount	DV	3.5.3
0x2C	IsProgrammable	DV	3.5.3
0x2D	InputBinding	DV	3.5.3
0x2E-4F	Reserved		
0x50	**LampMultiUpdateReport**	**CL**	3.5.4
0x51	RedUpdateChannel	DV	3.5.4
0x52	GreenUpdateChannel	DV	3.5.4
0x53	BlueUpdateChannel	DV	3.5.4
0x54	IntensityUpdateChannel	DV	3.5.4
0x55	LampUpdateFlags	DV	3.5.4
0x56-5F	Reserved		
0x60	**LampRangeUpdateReport**	**CL**	3.5.4
0x61	LampIdStart	DV	3.5.4
0x62	LampIdEnd	DV	3.5.4
0x63-6F	Reserved		
0x70	**LampArrayControlReport**	**CL**	3.5.5
0x71	AutonomousMode	DV	3.5.5
0x72-FFFF	Reserved		

Add to a new description section (per technical editor) after the Usage Page table with the following text

3 LampArray Operation

Typical LampArray operation has several phases;-

- Interrogation of LampArray device attributes.
- Interrogation of individual Lamp attributes.
- Disabling AutonomousMode on the device.
- Updating Lamp state.
- Enabling AutonomousMode on the device.

While it is not required that these phases are done in order (and no device should ever assume it), as we outline below, it should be clear that this is the most reasonable practice for a Host. See **Figure 1 Typical LampArray Operation**

Note: *Retrieval of Lamp colour state is not outlined since state is controlled exclusively by the Host which always knows the state it last set the device to. Future additions to this specification may include setting persistent state and its retrieval.*

Requirements

- Distance measurements to be given in micrometers (µm). For a signed 32bit integer (largest supported by HID), this gives a range from 1µm to >2km, which seems sufficient to describe any device.
- Time measurements to be given in microseconds (µs). For a signed 32bit integer (largest supported by HID), this gives a range from 1µs to >30minutes)

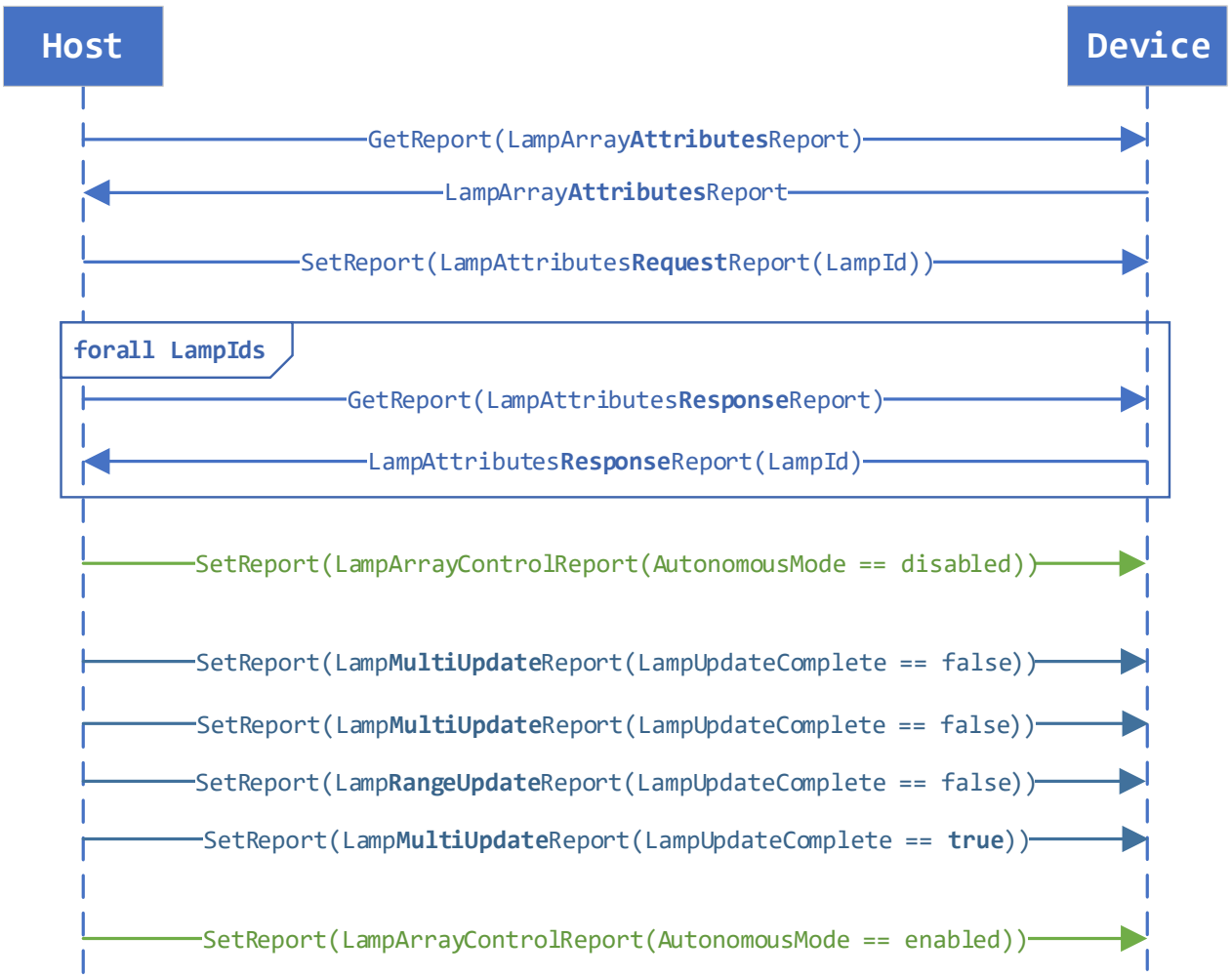

FIGURE 1 TYPICAL LAMPARRAY OPERATION

3.1 LampArray Attributes and Interrogation

Every `LampArray` is expected to have attributes describing the physical device that contains the `LampArray`. This includes the number of `Lamps` (`LampCount`), the kind of `LampArray` and dimensions of a logical bounding box. These values are static and can never change. `LampArrayAttributesReport` is used to retrieve these attributes.

`LampArrayKind` describes the type of physical device that contains the `LampArray` (e.g. keyboard/mouse/gamecontroller…). This helps the Host know what `Lamp` Attributes it can expect and associate it with other HID devices (keyboard/mouse). The kind must use one of `LampArrayKind*` values from the table in 3.6.1.

`BoundingBox*InMicrometers` describes a logical box encompassing the physical device. Origin (0,0,0) is that of the right-hand coordinate system (as prescribed in the HID spec 5.9 Orientation) which denotes the upmost, farthest, left-hand corner of the box. This box is used to provide the bounds of the device (without the detail/complexity of a true 3D model) and to provide a reference origin for `Lamp` coordinates. All sizes/coordinates/positions are thus positive offsets from this origin.

The dimensions and coordinate system is illustrated below with a typical keyboard and mouse in Figure 2, Figure 3 respectively.

In particular, notice:-

- Width is always perpendicular to the user when this keyboard/mouse is naturally orientated.
- Origin (0,0,0) is not flush with the corner of the keyboard as the device has a curved rise in the middle.
- Origin (0,0,0) is nowhere near the mouse body.

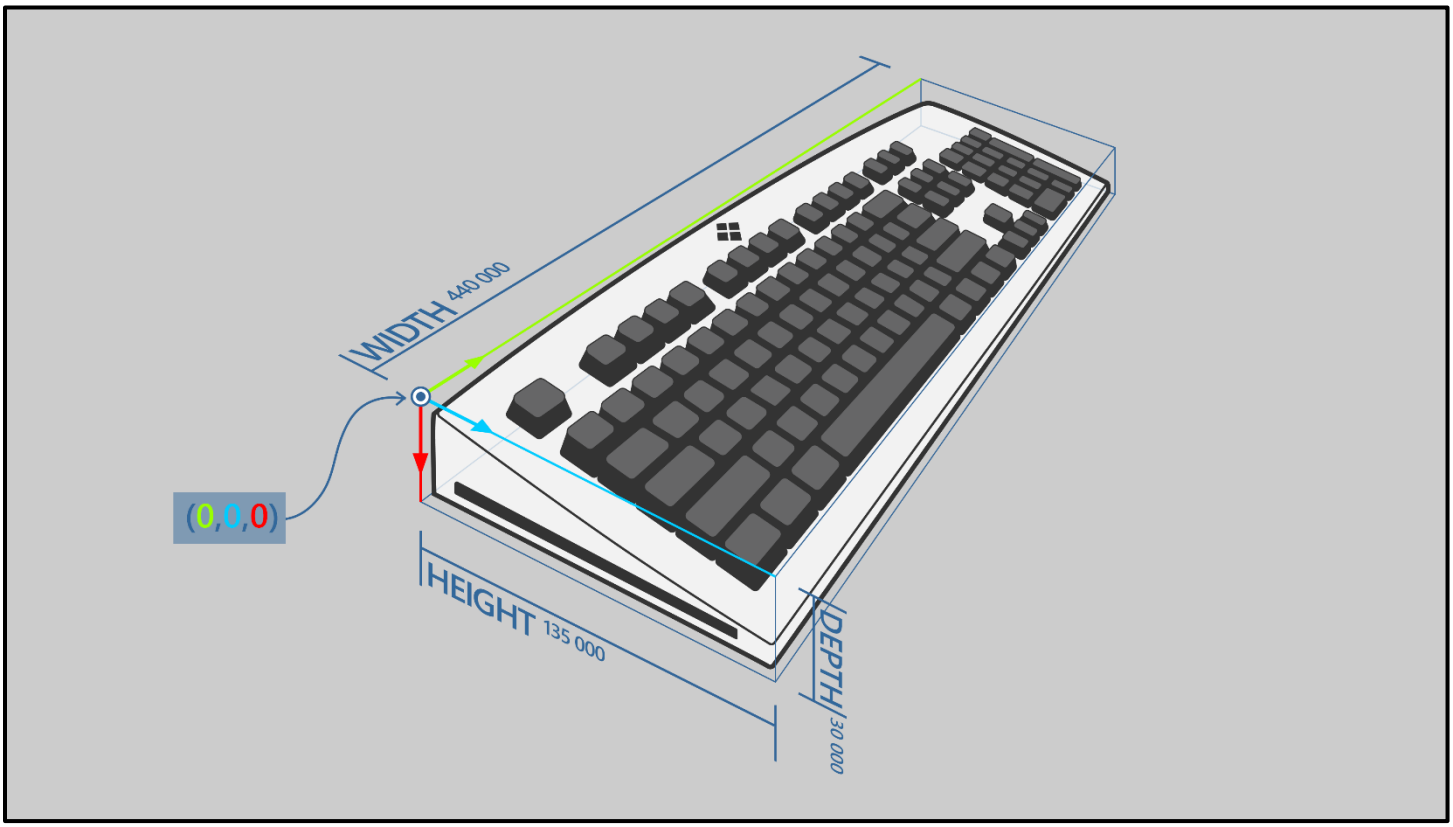

FIGURE 2 KEYBOARD WITH LABELED DIMENSIONS AND ORIGIN. LAMPS EXIST BENEATH EVERY KEY, THE BRANDING AT THE TOP/MIDDLE, AND ACCENT LIGHTING ALONG THE LEFT AND RIGHT SIDES. EXAMPLE SIZES GIVEN FOR EACH DIMENSION (IN MICROMETERS).

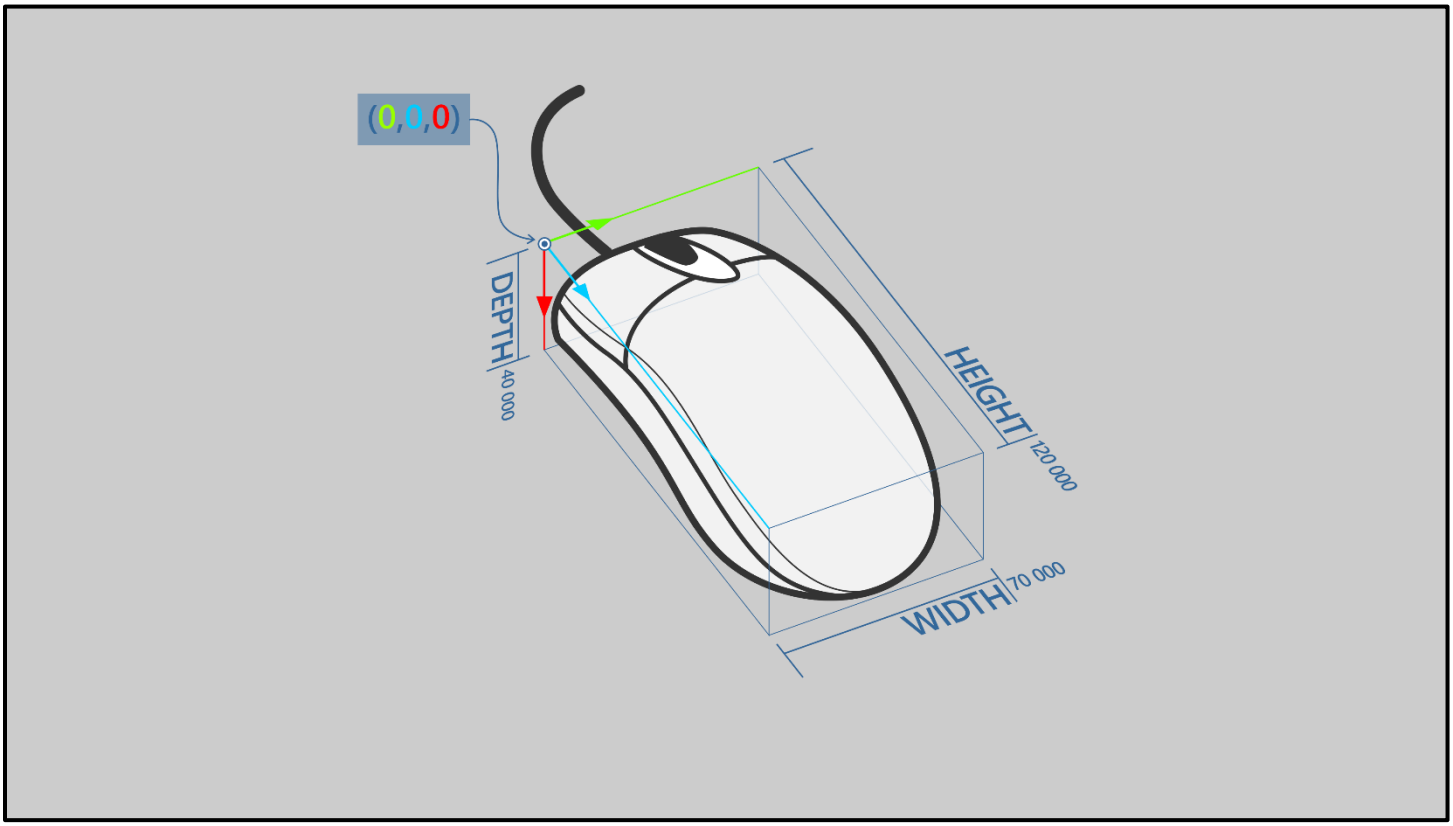

FIGURE 3 MOUSE WITH LABELED DIMENSIONS AND ORIGIN. EXAMPLE SIZES GIVEN FOR EACH DIMENSION (IN MICROMETERS).

`MinUpdateIntervalInMicroseconds` is the minimal time interval required for the Host to wait before sending two updates for any one `Lamp`. This is to prevent the Host overwhelming the device by sending too many `Lamp*UpdateReports` too quickly. A device must be able to accommodate updating *every* `Lamp` (individually) before requiring the Host to wait for the interval. This means a device must be able to receive and process (consecutively) the minimum number of `LampMultiUpdateReports` required to update all `Lamps`. This is so the Host knows it can update *every* `Lamp` on the device before waiting for the interval. If a Host misbehaves and sends more reports than allowed before waiting for the interval, the device can ignore those reports.

For example, a device where `LampArrayAttributes:LampCount==40` and `LampMultiUpdate:LampCount==8`, requires a minimum of (40/8=) 5 `LampMultiUpdateReports`; so 5 reports must be accepted before the Host is required to wait the interval.

3.2 Lamp Attributes and Interrogation

3.2.1 LampAttributesRequestReport

Having retrieved the LampCount, interrogation of a Lamp begins by the Host sending a LampAttributes**Request**Report (via Set Report) with the LampId of the first Lamp to interrogate. Each Lamp must have a unique LampId, numbered from 0 to LampCount-1 (inclusive). Lamps without a LampId cannot be referenced and must not be included in the LampArray. An invalid LampId, must be treated by the device as LampId==0.

It is recommended that LampIds are assigned to Lamps in a methodical manner (e.g. grid, starting from top-left) to take the most advantage of the LampRangeUpdateReport described below. This can significantly reduce traffic overhead of the update.

3.2.2 LampAttributesResponseReport

The Host then requests a LampAttributes**Response**Report (via GetReport) to which the device returns the attributes of the previously requested LampId.

Upon a successful response, the device will automatically (and internally) *increment* the previously sent LampId such that the next time the Host sends a LampAttributes**Response**Report, the device will return the attributes of the LampId+1 Lamp. Further requests monotonically increase the previous LampId.

After LampId==LampCount-1, the device will reset the internal LampId to 0, and continue to monotonically increase after each successful response. In this way a Host need only send a single LampAttributes**Request**Report for the first LampId to inspect (e.g. 0), then request multiple LampAttributes**Response**Reports; one for each further Lamp to inspect. Alternatively, a Host can explicitly send a LampAttributes**Request**Report before each LampAttributes**Response**Report instead of taking advantage of the automatic device increment; or a mix of the two patterns.

The default internal LampId is 0.

The Host must always check the LampId of the returned report to ensure it was expected (as an invalid LampId will always be treated as LampId==0)

3.2.2.1 Example

Sample operations of LampArray with 6 Lamps. Observe (#1-8) how the Host sets the LampId and then can receive multiple Response reports where the LampId increments by 1 each time until it resets to 0. Additionally see (#9-14) that the Host can still explicitly request which Lamp it should receive attributes for (e.g. if the Host wishes to request Lamps out of order).

#	ReportType	Direction	LampId
1	LampAttributes**Request**Report	OUT	0
2	LampAttributes**Response**Report	IN	0
3	LampAttributes**Response**Report	IN	1
4	LampAttributes**Response**Report	IN	2
5	LampAttributes**Response**Report	IN	3
6	LampAttributes**Response**Report	IN	4
7	LampAttributes**Response**Report	IN	5
8	LampAttributes**Response**Report	IN	0
9	LampAttributes**Request**Report	OUT	2
10	LampAttributes**Response**Report	IN	2
11	LampAttributes**Request**Report	OUT	4
12	LampAttributes**Response**Report	IN	4
13	LampAttributes**Response**Report	IN	5
14	LampAttributes**Response**Report	IN	0

3.2.3 Lamp Attributes

All `Lamp` attributes are static and can never change across device resets or external events.

`PositionX/Y/Z` describes the location of the `Lamp` (in 3D space) relative to the bounding-box origin defined in 3.1. Such data is useful for the Host when creating effects (e.g. animation moving from left to right). All `Lamps` are assumed to be a single, dimensionless point of zero size.

`LampPurposes` identifies the high-level purpose/s of the `Lamp`. This helps the Host determine what scenarios the Lamp can be used. The value must be composed of one or more `LampPurposes*` flags described in the table 3.6.2.

In the figures below we can see Lamps with different LampPurposes labeled with example X/Y/Z positions.

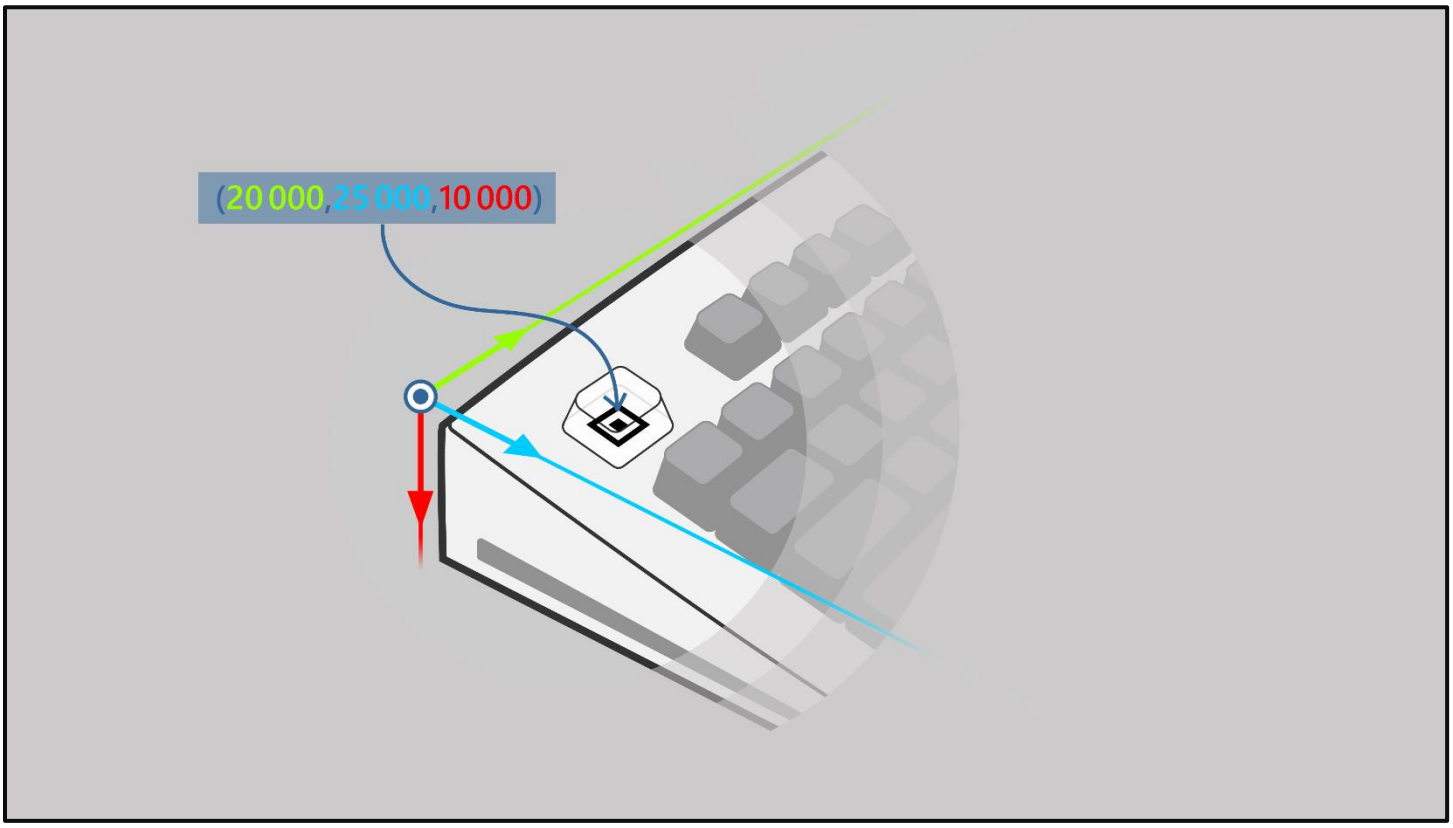

FIGURE 4 KEYBOARD CONTROL LAMP UNDER THE ESC KEY. POSITION OF LAMP RELATIVE TO THE BOUNDING BOX (DESCRIBED ABOVE) LABELED IN MICROMETERS.

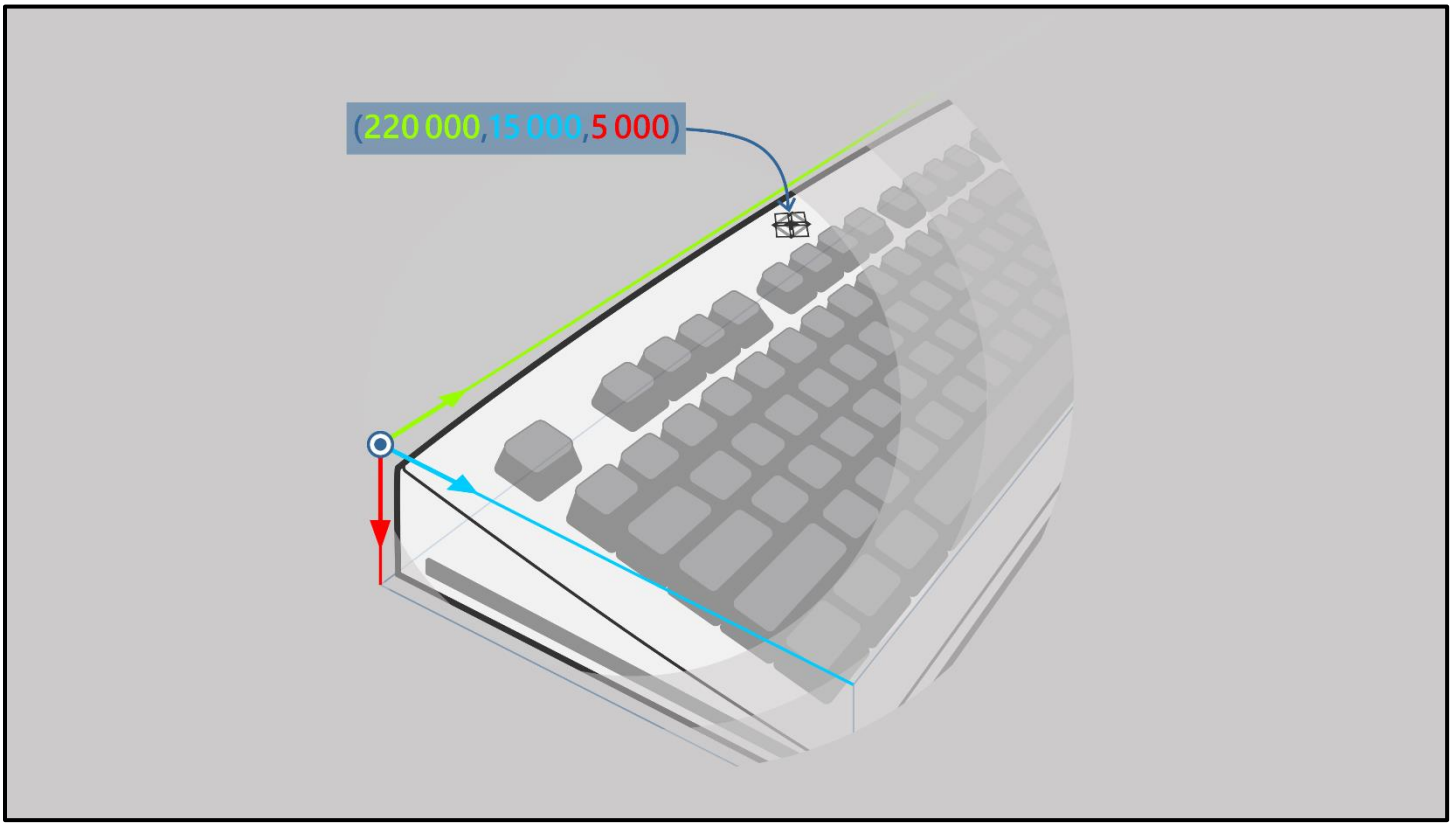

FIGURE 5 KEYBOARD BRANDING LAMP UNDER THE LOGO. POSITION OF LAMP RELATIVE TO THE BOUNDING BOX (DESCRIBED ABOVE) LABELED IN MICROMETERS.

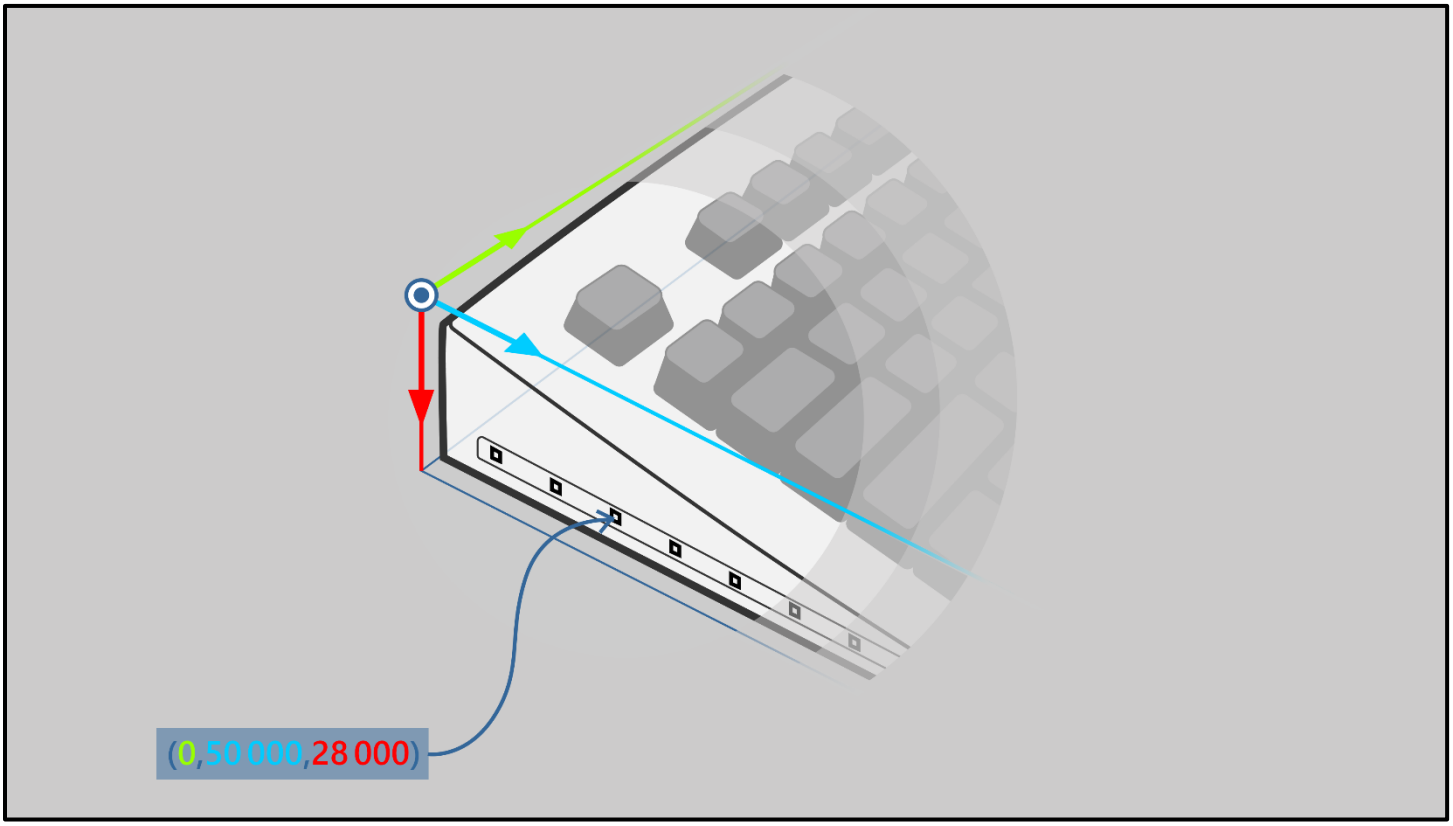

FIGURE 6 ACCENT LAMP ON THE LHS (PART OF ACCENT LIGHTING). POSITION OF LAMP RELATIVE TO THE BOUNDING BOX (DESCRIBED ABOVE) LABELED IN MICROMETERS.

`UpdateLatencyInMicroseconds` describes the smallest time interval between a device receiving a `Lamp*UpdateReport` and the state emanating from the device. This includes the time spent processing the report and the update latency of the specific `Lamp` (e.g. LEDs switch faster than incandescent lamps). This must be determined by the manufacturer and an upper bound given from a `Lamp` being completely off to any color intensity. This allows a Host to coordinate effects between multiple devices. It is expected (though not required) that this value will be identical for all `Lamps` of the same electrical/mechanical type on a device.

`InputBinding` associates a `Lamp` with either a keyboard/keypad key or a mouse button. This is to support today's common case of keyboards/mice with individually backlit keys/buttons. If the `LampArrayKind` declares the device as a keyboard, `InputBinding` must use one of the unsigned 16bit `Keyboard*` Usages from the Keyboard/Keypad UsagePage (`0x07`). If declared as a mouse, `InputBinding` must use one of the unsigned 16bit Button* Usages from the Button UsagePage (`0x09`) in the range of Button1 (`0x01`) to Button5 (`0x05`) inclusive. No more than 5 buttons are supported for any mouse.

If a key/button is not associated with this `Lamp` or it is not declared as either a keyboard or mouse, this value must be 0; non-zero values must be ignored by the Host.

3.2.4 Color Attributes

`LampArrays` support both `FixedColor` and `Programmable Lamps`. For programmable `Lamps` (indicated by `IsProgrammable==1`), `*LevelCount` values indicate the levels of intensity supported by the red, green, and blue color channels, each of which can be varied independently. Zero indicates an "off" state, and non-zero values indicate varying levels of color intensity. For example:

- A value of zero indicates that the color channel is not supported.
- A value of one indicates that the only intensities supported for the color channel are fully on and fully off.
- A value of 10 indicates that ten levels of intensity are supported, in addition to being turned off.

The highest non-zero intensity level corresponds to the maximum possible brightness for that color channel. Intensity values map as closely as possible to a visually linear brightness curve.

`IntensityLevelCount` indicates how many levels of *overall* intensity are supported for a `Lamp`. Zero indicates an "off" state, and non-zero values indicate varying levels of overall intensity/gain for a `Lamp`. Any number of intensity levels > 1 is supported. The highest non-zero intensity value corresponds to unity gain (maximum intensity), with intermediate values describing relative linear gain.

`FixedColor Lamps` (`IsProgrammable==0`) have a single fixed color at maximum intensity described by the relative color intensities of `RedLevelCount, GreenLevelCount, BlueLevelCount`. `IntensityLevelCount` can optionally be described (to vary overall intensity), but minimally most support 0 (off), 1 (on); intermediate intensity levels are scaled.

3.2.5 Color Attributes Examples

The table below illustrates examples of programmable `Lamps`, and how they are expressed via `RedLevelCount`, `GreenLevelCount`, `BlueLevelCount`, `IntensityLevelCount`.

Red	Green	Blue	Intensity	Meaning
1	1	0	1	A lamp that can be red, yellow, or green. The only intensities available are on or off.
1	0	1	32	A lamp that can be red, blue, or purple. The overall intensity of the lamp can be set to one of 32 levels, but the relative intensity of the red/blue channels cannot.
16	16	0	1	A red/green lamp that supports 256 unique colors.
255	255	255	1	An RGB lamp that supports 16,777,216 unique colors.

3.3 LampArrayControlReport

This report is defined to control various device-wide settings. All settings are non-persistent unless explicitly marked.

3.3.1 AutonomousMode

`AutonomousMode` is a boolean field indicating whether the device can decide whether/how to update `Lamps` itself, or if the Host has the exclusive ability to set/update the `Lamp` state. No source other than the Host can modify `Lamp` state while field is disabled/false. When enabled/true, the `Lamp` state can be set by other sources (e.g. `Lamp` state set manually on device or reverts to an embedded default effect) and any `Lamp*UpdateReports` can be ignored. Default state for this field is enabled/true. The device must always handle `LampArrayAttributesReports`, `LampAttributesRequestReports`, `LampAttributesResponseReports`, regardless of this field state.

When disabled, only the Host may change the `Lamp` state (via `Lamp*UpdateReports`). Once disabled (and was previously enabled, but before sending `Lamp*UpdateReports`) the device must 'pause' any playing effect it started and maintain `Lamps` to whatever was last set by the device (e.g. if displaying solid blue in autonomous mode, once disabled, solid blue must persist). If the field was previously enabled (and is set to enabled again), it is a no-op. Similarly, disabling when already disabled is a no-op.

After the field is disabled, sent `Lamp*UpdateReports` will change the `Lamp` state from the last device-set state. It is up to the Host to 'override' the persisted state by sending `Lamp*UpdateReports`. The Host with guarantee to wait for `MinUpdateIntervalInMicroseconds` before sending its first `Lamp*UpdateReport`.

If this field is absent, it means no autonomous mode is supported. If supported, the device should default to enabled/true.

3.4 Updating Lamp State

Two reports are defined (Lamp**Multi**UpdateReport, Lamp**Range**UpdateReport) to accommodate expected classes of updates. Both updates are non-persistent, such that if a device loses power, or is moved to a different Host, the Lamp returns to it's default state. Default state for all Lamps is 'off' (RGBI=0,0,0,0).

Update reports can contain flags (LampUpdateFlags) to describe the update; currently, LampUpdateComplete is the only valid flag. LampUpdateComplete is set by the Host when the report is the last update in a batch of updates, and the device should alter the Lamp states all at once. Devices can wait until an update with this flag has been received before applying any of the previous updates. The Host guarantees to not send more than 1 update with this flag every MinUpdateIntervalInMicroseconds.

3.4.1 LampMultiUpdateReport

Lamp**Multi**UpdateReport updates the color of multiple Lamps in a single request, where all four channels (Red/Green/Blue/Intensity) can be set at once for given Lamps. The MaxLogicalSize of the LampCount Usage in the descriptor defines the number of available update-slots. Within the report, LampCount identifies the number of update slots to be examined (starting from the first slot). LampIds do not have to be ordered (e.g. ascending), but update-slot position identifies corresponding RGBI tuple. Update-slots must always be filled from 0 to max(LampCount). Any unused slot must be ignored by the device. (It is recommended the Host set both the LampId and the corresponding channel intensities to 0).

For FixedColor Lamps, only the Intensity channel is examined by the device (i.e. Red/Green/Blue channels are always ignored; as a best practice these channels should always be set to 0 by the Host).

If any error is detected by the device in the report, the device shall ignore the entire report. Errors include:-

- Any LampId >= Device LampCount
- *Channel > *LevelCount described by Lamps' attributes. (e.g. if a Lamp had RedLevelCount==100 and an update set the channel to 101)
- Identical LampId in multiple slots.

In the example below is a LampMultiUpdateReport which has 8 update slots and declares it has 5 Lamps to update; slots 6/7/8 are hence ignored by the device. " LampId#1" (0x19) corresponds to "RGBI tuple #1" (FF 00 FF 80) etc…

LampCount	0x05
LampId #1	0x19
LampId #2	0x23
LampId #3	0x72
LampId #4	0x56
LampId #5	0x64
LampId #6	(ignored)
LampId #7	(ignored)
LampId #8	(ignored)
RGBI tuple #1	0xFF 0x00 0xFF 0x80
RGBI tuple #2	0x80 0x80 0xFF 0xFF
RGBI tuple #3	0x00 0x00 0x80 0xFF
RGBI tuple #4	0xFF 0x80 0x00 0x80
RGBI tuple #5	0xFF 0xFF 0x00 0xFF
RGBI tuple #6	(ignored)
RGBI tuple #7	(ignored)
RGBI tuple #8	(ignored)

3.4.2 LampRangeUpdateReport

Lamp**Range**UpdateReport allows multiple Lamps to be updated based on the range between two LampIds. LampIdStart and LampIdEnd are both included in the range. The single Red/Green/Blue/Intensity color is applied to every Lamp within the range. A common use-case for range to turn all Lamps 'off' (LampIdStart==0, LampIdEnd==(LampCount-1), RGBI==0).

For FixedColor Lamps, Red/Green/Blue channels are always ignored.

If any error is detected by the device in the report, the device shall ignore the entire report. Errors include:-

- LampIdStart > LampIdEnd
- LampIdStart || LampIdEnd >= Device LampCount
- *Channel > *LevelCount described by any Lamps' attributes within the range. (i.e. The Host must ensure all Lamps in the described range support the desired channel intensities)

FixedColor Lamps may be mixed with Programmable Lamps within the range so long as the desired IntensityUpdateChannel is within range. RGB channels for FixedColor Lamps will be ignored.

In the example below, all Lamps between (and including) LampId=0x19 to LampId=0x31 are set to the corresponding RGBI value (0xFF 0x00 0x00 0xFF).

LampIdStart	0x19
LampIdEnd	0x31
RGBI tuple	0xFF 0x00 0x00 0xFF

3.5 Usage Definitions

3.5.1 Application Usages

Usage Name	Description
LampArray	CA - Applied to a collection containing LampArray attributes and reports.

3.5.2 LampArray Attributes Report

Usage Name	Description
LampArrayAttributesReport	CL – Feature – Applied to a collection containing the device attributes of a LampArray
LampCount	SV/DV – Feature – Number of Lamps associated with a LampArray
BoundingBoxWidthInMicrometers	SV – Feature – Width (X axis) of a logical bounding-box encompassing the device. Must be a positive offset from the origin.
BoundingBoxHeightInMicrometers	SV – Feature – Height (Y axis) of a logical bounding-box encompassing the device. Must be a positive offset from the origin.
BoundingBoxDepthInMicrometers	SV – Feature – Depth (Z axis) of a logical bounding-box encompassing the device. Must be a positive offset from the origin.
LampArrayKind	SV – Feature – Kind of LampArray. Must be one of the values defined in 3.6.1
MinUpdateIntervalInMicroseconds	SV – Feature – Minimal time interval required between the Host sending two updates for any one Lamp.

3.5.3 Lamp Attributes Report

Usage Name	Description
LampAttributesRequestReport	CL – Feature – Applied to a collection containing a LampId to request attributes for
LampAttributesResponseReport	CL – Feature – Applied to a collection containing attributes corresponding to a requested LampId
LampId	SV/DV – Feature – Id of a Lamp. Valid range is between 0 and (LampCount – 1)
PositionXInMicrometers	DV – Feature – X position (corresponding to Bounding Box Width) from origin
PositionYInMicrometers	DV – Feature – Y position (corresponding to Bounding Box Height) from origin
PositionZInMicrometers	DV – Feature – Z position (corresponding to Bounding Box Depth) from origin
LampPurposes	DV – Feature – Purpose/s of a Lamp. Must be one or more flags from table in 3.6.2
UpdateLatencyInMicroseconds	DV – Feature – Time interval between the device receiving an update for a Lamp and it emanating from the device.
RedLevelCount	DV – Feature – The number of red color intensities settable for this LampId.
GreenLevelCount	DV – Feature – The number of green color intensities settable for this LampId.
BlueLevelCount	DV – Feature – The number of blue color intensities settable for this LampId.
IntensityLevelCount	DV – Feature – The number of color independent intensities settable for this LampId.
IsProgrammable	DV – Feature – 1 if this Lamp has programmable colors, 0 if it doesn't.
InputBinding	DV – Feature –Keyboard* Usages from the Keyboard/Keypad UsagePage (0x07) or Button* Usages from Button UsagePage (0x09)

3.5.4 Lamp Update Reports

Usage Name	Description
LampMultiUpdateReport	CL – Feature/Output – Applied to a collection containing updates for multiple Lamps, each Lamp specified can have a different color.
LampRangeUpdateReport	CL – Feature/ Output – Applied to a collection containing a single range update consisting of color channels and LampIdStart/LampIdEnd. All Lamps within range are set to the same color.
LampUpdateFlags	DV – Feature/Output – Flags associated with a Lamp*Update message. See table 3.6.3
RedUpdateChannel	DV – Feature/Output – Red intensity of the new color for this LampId. Ignored unless Lamp IsProgrammable is true.
GreenUpdateChannel	DV – Feature/Output – Green intensity of the new color for this LampId. Ignored unless Lamp IsProgrammable is true.
BlueUpdateChannel	DV – Feature/Output – Blue intensity of the new color for this LampId. Ignored unless Lamp IsProgrammable is true.
IntensityUpdateChannel	DV – Feature/Output – Intensity/gain overall of the new color for this LampId.
LampIdStart	DV – Feature/Output – The first LampId in the range of LampIds to update
LampIdEnd	DV – Feature/Output – The last LampId in the range of LampIds to update

3.5.5 LampArray Control Report

Usage Name	Description
LampArrayControlReport	CL – Feature – Applied to a collection containing LampArray control fields
AutonomousMode	DV – Feature – Boolean value indicating whether the device can set Lamp state itself/autonomously (i.e. without the Host sending Lamp update messages). Default value is enabled/true.

3.6 Enumeration Definitions

3.6.1 LampArrayKind Value Table

Name	Description	Value
Undefined	*Undefined*	0x00
LampArrayKindKeyboard	LampArray is part of a keyboard/keypad device	0x01
LampArrayKindMouse	LampArray is part of a mouse	0x02
LampArrayKindGameController	LampArray is part of a game-controller. (e.g. gamepad, flightstick, sailing simulation device)	0x03
LampArrayKindPeripheral	LampArray is part of a general peripheral/accessory (e.g. speakers, mousepad, microphone, webcam)	0x04
LampArrayKindScene	LampArray illuminates a room/performance-stage/area (e.g. room light-bulbs, spotlights, washlights, strobelights, booth-strips, billboard/sign, camera-flash)	0x05
LampArrayKindNotification	LampArray is part of a notification device	0x06
LampArrayKindChassis	LampArray is part of an internal PC case component (e.g. RAM-stick, motherboard, fan)	0x07
LampArrayKindWearable	LampArray is embedded in a wearable accessory (audio-headset, wristband, watch, shoes)	0x08
LampArrayKindFurniture	LampArray is embedded in a piece of funiture (e.g. chair, desk, bookcase)	0x09
LampArrayKindArt	LampArray is embedded in an artwork (e.g. painting, sculpture)	0x0A
Reserved	*Reserved*	0x0B-0xFFFF
Vendor-Defined	Vendor-Defined	0x10000-0xFFFF**FFFF**

3.6.2 LampPurposes Flags Table

Note: Flags are permitted to be combined. Lacking any flags for this field (i.e. 0x00) is undefined.

Name	Description	Flag
LampPurposeControl	Control Lamp (e.g. button/key/slider etc…)	0x01
LampPurposeAccent	Accent Lamp that doesn't interact with the user (e.g. case fan LED, illuminated side panels on a keyboard)	0x02
LampPurposeBranding	Device branding (e.g. Manufacturer logo)	0x04
LampPurposeStatus	Status Lamp (e.g. unread email, CPU temperature)	0x08
LampPurposeIllumination	Illuminates an object that is outside of the LampArray (e.g. stage spotlight, car headlights, camera flash)	0x10
LampPurposePresentation	A Lamp the user directly looks at (e.g. within an artwork or costume)	0x20
Reserved	*Reserved*	0x40-0xFFFF
Vendor-Defined	Vendor-Defined	0x10000-0xFFFF**FFFF**

3.6.3 LampUpdateFlags Table

Note: Flags are permitted to be combined.

Name	Description	Flag
LampUpdateComplete	Signals that this was the last update in a batch of updates. Device should now process all precceding messages as a single update to Lamp state.	0x01
Reserved	*Reserved*	0x02-0xFFFF

4 Example Descriptor

```
0x05, 0x59,                    // USAGE_PAGE (LightingAndIllumination)
0x09, 0x01,                    // USAGE (LampArray)
0xa1, 0x01,                    // COLLECTION (Application)
0x85, 0x01,                    //   REPORT_ID (1)
0x09, 0x02,                    //   USAGE (LampArrayAttributesReport)
0xa1, 0x02,                    //   COLLECTION (Logical)
0x09, 0x03,                    //     USAGE (LampCount)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,  //     LOGICAL_MAXIMUM (65535)
0x75, 0x10,                    //     REPORT_SIZE (16)
0x95, 0x01,                    //     REPORT_COUNT (1)
0xb1, 0x03,                    //     FEATURE (Cnst,Var,Abs)
0x09, 0x04,                    //     USAGE (BoundingBoxWidthInMicrometers)
0x09, 0x05,                    //     USAGE (BoundingBoxHeightInMicrometers)
0x09, 0x06,                    //     USAGE (BoundingBoxDepthInMicrometers)
0x09, 0x07,                    //     USAGE (LampArrayKind)
0x09, 0x08,                    //     USAGE (MinUpdateIntervalInMicroseconds)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0xff, 0x7f,  //     LOGICAL_MAXIMUM (2147483647)
0x75, 0x20,                    //     REPORT_SIZE (32)
0x95, 0x05,                    //     REPORT_COUNT (5)
0xb1, 0x03,                    //     FEATURE (Cnst,Var,Abs)
0xc0,                          //   END_COLLECTION
0x85, 0x02,                    //   REPORT_ID (2)
0x09, 0x20,                    //   USAGE (LampAttributesRequestReport)
0xa1, 0x02,                    //   COLLECTION (Logical)
0x09, 0x21,                    //     USAGE (LampId)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,  //     LOGICAL_MAXIMUM (65535)
0x75, 0x10,                    //     REPORT_SIZE (16)
0x95, 0x01,                    //     REPORT_COUNT (1)
0xb1, 0x02,                    //     FEATURE (Data,Var,Abs)
0xc0,                          //   END_COLLECTION
0x85, 0x03,                    //   REPORT_ID (3)
0x09, 0x22,                    //   USAGE (LampAttributesReponseReport)
0xa1, 0x02,                    //   COLLECTION (Logical)
0x09, 0x21,                    //     USAGE (LampId)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,  //     LOGICAL_MAXIMUM (65535)
0x75, 0x10,                    //     REPORT_SIZE (16)
0x95, 0x01,                    //     REPORT_COUNT (1)
0xb1, 0x02,                    //     FEATURE (Data,Var,Abs)
0x09, 0x23,                    //     USAGE (PositionXInMicrometers)
0x09, 0x24,                    //     USAGE (PositionYInMicrometers)
0x09, 0x25,                    //     USAGE (PositionZInMicrometers)
0x09, 0x27,                    //     USAGE (UpdateLatencyInMicroseconds)
0x09, 0x26,                    //     USAGE (LampPurposes)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0xff, 0x7f,  //     LOGICAL_MAXIMUM (2147483647)
0x75, 0x20,                    //     REPORT_SIZE (32)
```

```
0x95, 0x05,                        //      REPORT_COUNT (5)
0xb1, 0x02,                        //      FEATURE (Data,Var,Abs)
0x09, 0x28,                        //      USAGE (RedLevelCount)
0x09, 0x29,                        //      USAGE (GreenLevelCount)
0x09, 0x2a,                        //      USAGE (BlueLevelCount)
0x09, 0x2b,                        //      USAGE (IntensityLevelCount)
0x09, 0x2c,                        //      USAGE (IsProgrammable)
0x09, 0x2d,                        //      USAGE (InputBinding)
0x15, 0x00,                        //      LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,                  //      LOGICAL_MAXIMUM (255)
0x75, 0x08,                        //      REPORT_SIZE (8)
0x95, 0x06,                        //      REPORT_COUNT (6)
0xb1, 0x02,                        //      FEATURE (Data,Var,Abs)
0xc0,                              //    END_COLLECTION
0x85, 0x04,                        //    REPORT_ID (4)
0x09, 0x50,                        //    USAGE (LampMultiUpdateReport)
0xa1, 0x02,                        //    COLLECTION (Logical)
0x09, 0x03,                        //      USAGE (LampCount)
0x09, 0x55,                        //      USAGE (LampUpdateFlags)
0x15, 0x00,                        //      LOGICAL_MINIMUM (0)
0x25, 0x08,                        //      LOGICAL_MAXIMUM (8)
0x75, 0x08,                        //      REPORT_SIZE (8)
0x95, 0x02,                        //      REPORT_COUNT (2)
0xb1, 0x02,                        //      FEATURE (Data,Var,Abs)
0x09, 0x21,                        //      USAGE (LampId)
0x15, 0x00,                        //      LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,      //      LOGICAL_MAXIMUM (65535)
0x75, 0x10,                        //      REPORT_SIZE (16)
0x95, 0x08,                        //      REPORT_COUNT (8)
0xb1, 0x02,                        //      FEATURE (Data,Var,Abs)
0x09, 0x51,                        //      USAGE (RedUpdateChannel)
0x09, 0x52,                        //      USAGE (GreenUpdateChannel)
0x09, 0x53,                        //      USAGE (BlueUpdateChannel)
0x09, 0x54,                        //      USAGE (IntensityUpdateChannel)
0x09, 0x51,                        //      USAGE (RedUpdateChannel)
0x09, 0x52,                        //      USAGE (GreenUpdateChannel)
0x09, 0x53,                        //      USAGE (BlueUpdateChannel)
0x09, 0x54,                        //      USAGE (IntensityUpdateChannel)
0x09, 0x51,                        //      USAGE (RedUpdateChannel)
0x09, 0x52,                        //      USAGE (GreenUpdateChannel)
0x09, 0x53,                        //      USAGE (BlueUpdateChannel)
0x09, 0x54,                        //      USAGE (IntensityUpdateChannel)
0x09, 0x51,                        //      USAGE (RedUpdateChannel)
0x09, 0x52,                        //      USAGE (GreenUpdateChannel)
0x09, 0x53,                        //      USAGE (BlueUpdateChannel)
0x09, 0x54,                        //      USAGE (IntensityUpdateChannel)
0x09, 0x51,                        //      USAGE (RedUpdateChannel)
0x09, 0x52,                        //      USAGE (GreenUpdateChannel)
0x09, 0x53,                        //      USAGE (BlueUpdateChannel)
0x09, 0x54,                        //      USAGE (IntensityUpdateChannel)
0x09, 0x51,                        //      USAGE (RedUpdateChannel)
0x09, 0x52,                        //      USAGE (GreenUpdateChannel)
0x09, 0x53,                        //      USAGE (BlueUpdateChannel)
```

```
0x09, 0x54,                    //     USAGE (IntensityUpdateChannel)
0x09, 0x51,                    //     USAGE (RedUpdateChannel)
0x09, 0x52,                    //     USAGE (GreenUpdateChannel)
0x09, 0x53,                    //     USAGE (BlueUpdateChannel)
0x09, 0x54,                    //     USAGE (IntensityUpdateChannel)
0x09, 0x51,                    //     USAGE (RedUpdateChannel)
0x09, 0x52,                    //     USAGE (GreenUpdateChannel)
0x09, 0x53,                    //     USAGE (BlueUpdateChannel)
0x09, 0x54,                    //     USAGE (IntensityUpdateChannel)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,              //     LOGICAL_MAXIMUM (255)
0x75, 0x08,                    //     REPORT_SIZE (8)
0x95, 0x20,                    //     REPORT_COUNT (32)
0xb1, 0x02,                    //     FEATURE (Data,Var,Abs)
0xc0,                          //   END_COLLECTION
0x85, 0x05,                    //   REPORT_ID (5)
0x09, 0x60,                    //   USAGE (LampRangeUpdateReport)
0xa1, 0x02,                    //   COLLECTION (Logical)
0x09, 0x55,                    //     USAGE (LampUpdateFlags)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x25, 0x08,                    //     LOGICAL_MAXIMUM (8)
0x75, 0x08,                    //     REPORT_SIZE (8)
0x95, 0x01,                    //     REPORT_COUNT (1)
0xb1, 0x02,                    //     FEATURE (Data,Var,Abs)
0x09, 0x61,                    //     USAGE (LampIdStart)
0x09, 0x62,                    //     USAGE (LampIdEnd)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x27, 0xff, 0xff, 0x00, 0x00,  //     LOGICAL_MAXIMUM (65535)
0x75, 0x10,                    //     REPORT_SIZE (16)
0x95, 0x02,                    //     REPORT_COUNT (2)
0xb1, 0x02,                    //     FEATURE (Data,Var,Abs)
0x09, 0x51,                    //     USAGE (RedUpdateChannel)
0x09, 0x52,                    //     USAGE (GreenUpdateChannel)
0x09, 0x53,                    //     USAGE (BlueUpdateChannel)
0x09, 0x54,                    //     USAGE (IntensityUpdateChannel)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x26, 0xff, 0x00,              //     LOGICAL_MAXIMUM (255)
0x75, 0x08,                    //     REPORT_SIZE (8)
0x95, 0x04,                    //     REPORT_COUNT (4)
0xb1, 0x02,                    //     FEATURE (Data,Var,Abs)
0xc0,                          //   END_COLLECTION
0x85, 0x06,                    //   REPORT_ID (6)
0x09, 0x70,                    //   USAGE (LampArrayControlReport)
0xa1, 0x02,                    //   COLLECTION (Logical)
0x09, 0x71,                    //     USAGE (AutonomousMode)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x25, 0x01,                    //     LOGICAL_MAXIMUM (1)
0x75, 0x08,                    //     REPORT_SIZE (8)
0x95, 0x01,                    //     REPORT_COUNT (1)
0xb1, 0x02,                    //     FEATURE (Data,Var,Abs)
0xc0,                          //   END_COLLECTION
0xc0                           // END_COLLECTION
```