Integrating Ambient Light Sensors with Computers Running Windows 10:

Considerations for Interfacing Ambient Light Sensors and implementing adaptive brightness in Windows.

March 25, 2019

Abstract

This paper provides information about ambient light sensor (ALS) features in the Windows 10 operating system. It provides guidelines for hardware manufacturers and original equipment manufacturers (OEMs) to integrate ALS hardware with computers that use the adaptive brightness feature and with applications that use light sensor data.

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the Web at:   
 [Integrating Ambient Light Sensors with Computers Running Windows 10](https://docs.microsoft.com/windows-hardware/design/whitepapers/integrating-ambient-light-sensors-with-computers-running-windows-10-creators-update)

**Disclaimer**: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice. Some information relates to pre-released product which may be substantially modified before it’s commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2019 Microsoft. All rights reserved.

C:\Users\jenlin\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Outlook\KN5ONHWU\dep_MicrosoftLogotype.png

Document History

| **Date** | **Change** |
| --- | --- |
| March 25, 2019 | Updates for Windows 10 version 1903. |
| August 29, 2018 | Updates for alignment with Windows 10 version 1809. Included considerations for nits display support. Section reordering for increased clarity. Added test and validation procedures. |
| February 20, 2018 | Updated links to external documents. |
| May 18, 2019 | Updates for clarity and consistency. |
| April 18, 2019 | Updates for alignment with Windows 10 Creators Update RTM. |
| March 22, 2019 | Include sample ALR curve and sample lux-to-nits dataset. |
| March 6, 2019 | Corrected registry key path and name. Add information around upgrade behavior from Windows 10 Anniversary Update and early releases. |
| February 13, 2019 | Updates including sample ALR curve and clarification of required registry keys. |
| January 31, 2019 | Updates for alignment with Windows 10 Creators Update. |
| September 28, 2012 | Updates to reflect rebranding; updated Light Sensor Features, Ambient Light Sensor Integration Checklist, Understanding Adaptive Brightness Support in Windows 8, Integrating Light Sensors with Computer Hardware, and ALR Curve Type. |
| July 31, 2012 | Updates for alignment with Windows 8 RTM. |
| June 13, 2012 | First publication for Windows 8. |

Contents

[Terminology 5](#_Toc8914110)

[New to Windows 10 version 1903 6](#_Toc8914111)

[Understanding Adaptive Brightness Support in Windows 10 version 1903 7](#_Toc8914112)

[Simplified Windows 10 Adaptive Brightness System Architecture 7](#_Toc8914113)

[Adaptive Brightness Data Flow 8](#_Toc8914114)

[Brightness Setting and UI 9](#_Toc8914115)

[Removal of brightness slider value differences between AC and DC mode 11](#_Toc8914116)

[Registry Configuration Parameters 12](#_Toc8914117)

[Light Sensor Features 13](#_Toc8914118)

[Automatic Screen Brightness (Adaptive Brightness) 13](#_Toc8914119)

[Battery Life Considerations 14](#_Toc8914120)

[Automatic Keyboard Illumination 14](#_Toc8914121)

[Light-Aware Applications 14](#_Toc8914122)

[Ambient Light Sensor Integration Checklist 15](#_Toc8914123)

[Integrating Light Sensors with Device Hardware 16](#_Toc8914124)

[Types of ambient light sensors 17](#_Toc8914125)

[Number of Light Sensors 17](#_Toc8914126)

[Light Sensor Placement 18](#_Toc8914127)

[Dealing with invalid light sensor data 19](#_Toc8914128)

[Light Sensor Filters, Lenses, Enclosures, and Calibration 19](#_Toc8914129)

[Sensor Connectivity with HID and SPB 22](#_Toc8914130)

[Ambient Light Sensor Calibration 24](#_Toc8914131)

[Professional Calibration (Recommended) 24](#_Toc8914132)

[Other Calibration Techniques 24](#_Toc8914133)

[Optimizing Display Brightness Steps and Transitions 25](#_Toc8914134)

[Brightness and display considerations 25](#_Toc8914135)

[Controlling brightness using nits values 25](#_Toc8914136)

[Controlling brightness using percentage values 26](#_Toc8914137)

[ACPI considerations 28](#_Toc8914138)

[Tuning and Configuring Adaptive Brightness Behavior 28](#_Toc8914139)

[Lux-to-Nits Relationship 30](#_Toc8914140)

[Ambient Light Response (ALR) Curve 31](#_Toc8914141)

[Understanding Ambient Light Response (ALR) Curve Data 32](#_Toc8914142)

[Constructing a High-Fidelity Ambient Light Response (ALR) Curve from User Preference Data 34](#_Toc8914143)

[Registry settings 35](#_Toc8914144)

[Registry settings for **Percentage based** brightness 35](#_Toc8914145)

[ALR Curve Setting (ALRPoints) 35](#_Toc8914146)

[ALR Curve Version Setting (ALRCurveVersion) 36](#_Toc8914147)

[Registry settings for **Nits based** brightness 37](#_Toc8914148)

[Registry location 37](#_Toc8914149)

[Tests and validation 37](#_Toc8914150)

[Validation tools 37](#_Toc8914151)

[Display validation 38](#_Toc8914152)

[Light sensor validation 38](#_Toc8914153)

[Test cases 40](#_Toc8914154)

[Conclusion 51](#_Toc8914155)

[Additional documentation 51](#_Toc8914156)

[Sensor Driver Requirements 51](#_Toc8914157)

[Resources 51](#_Toc8914158)

# Terminology

| Term | Definition |
| --- | --- |
| ABS | Automatic brightness service |
| ALS | Ambient light sensor |
| ALR | Ambient light response |
| SPB | Simple peripheral bus (low power bus such as I2C) |
| Light pipe | A light directing device (such as a clear plastic extrusion) |
| HLK | Windows Hardware Lab Kit (formerly known as HCK) |
| HLK Test | A test that validates device firmware or drivers (device test) or a test that validates system-level features (system test) |
| Adaptive brightness (automatic brightness) | Automatic screen brightness control using an ambient light sensor in Windows.  Note: “Automatic brightness” and “Adaptive brightness” have an identical meaning and are used interchangeably in this document. |
| ALR curve | Ambient light response curve: a collection of points that defines the relationship between light levels (LUX) and offsets that are applied to a user’s screen brightness preference for adaptive brightness |
| Nit | Unit of luminance: 1 nit = 1 candela per square meter. Quantifies the brightness output of a display. |
| Lux | Unit of illuminance: 1 lux = 1 lumen per square meter. Quantifies the brightness of the environment; measured by an ALS sensor. |

# New to Windows 10 version 1903

This paper outlines Windows 10 version 1903 automatic brightness service updates. The content of this document is especially relevant to manufacturers shipping new machines with 1903. Customer machines upgraded from the Windows 10 version 1809 or earlier will continue to use brightness behaviors that were supported in earlier versions of the operating system. The Windows 10 version 1903 implementation improves on the version 1809 implementation.

Key changes:

* Adaptive Color support was added in Windows 10 version 1903. Please refer to this document on [Collaborate](https://partner.microsoft.com/en-us/dashboard/collaborate/packages/6943).
* New brightness slider was added to the quick action center.
* UI controls within Windows merge the AC/DC slider values

The [DXGK\_BRIGHTNESS\_INTERFACE\_3](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/dispmprt/nc-dispmprt-dxgk_brightness_set_3) interface introduced in 1809 and recommended in 1903 is optional. Existing systems can still rely on the DXGK\_BRIGHTNESS\_INTERFACE\_2 to control brightness. That said, the benefits brought by the [DXGK\_BRIGHTNESS\_INTERFACE\_3](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/dispmprt/nc-dispmprt-dxgk_brightness_set_3) interface are multifold:

* The DXGK\_BRIGHTNESS\_INTERFACE\_3 interface supports up to 4294967296 levels of brightness, in 1 millinit increments. Systems that support the DXGK\_BRIGHTNESS\_INTERFACE\_3, benefit from smooth, non-distracting brightness transitions.
* Systems implementing the DXGK\_BRIGHTNESS\_INTERFACE\_3 do not need to provide an ALR curve. The inbox Lux to Nits curve is designed to satisfy most devices. We will continue to tune this curve using satisfaction data, so that it continues to improve over time. It is worth noting that systems implementing the DXGK\_BRIGHTNESS\_INTERFACE\_3 must expose a well-calibrated display. If the system supports an ambient light sensor for automatic brightness, the sensor must be well-calibrated.
* Properly calibrated displays should look identical in terms of brightness for the same configuration.
* IHVs and OEMs can optionally offload brightness transitions to the underlying hardware. Doing so provides some battery life gains.

# Understanding Adaptive Brightness Support in Windows 10 version 1903

In Windows 10 version 1903, automatic brightness is enabled by default.

The following adaptive brightness components are included in Windows 10 version 1903:

* In-box class drivers for ambient light sensors
* HID Sensor Driver: A HID sensor class driver for an ALS that is connected behind a sensor microcontroller (MCU, or has HID support onboard the ALS chip).
* A service that consumes light sensor data and modifies display brightness accordingly.
* Settings in the System>Display page of the Windows Settings app to enable or disable adaptive brightness and modify the current screen brightness.
* Users can control brightness using the action center slider.
* Adaptive brightness registry configuration parameters.

The following subsections examine how these components support adaptive brightness.

## Simplified Windows 10 Adaptive Brightness System Architecture



Figure 1: Windows 10 Adaptive Brightness System Architecture

## Adaptive Brightness Data Flow

When an ALS is used for automatic screen brightness control, the following steps take place each time ambient light sensor data is updated:

1. The sensor driver raises an ALS data updated event.
2. The sensor service receives the ALS data event and compares the new data to the old data and timestamp. If a sustained change in lighting conditions is detected, a new brightness value (either percentage or nits value) is calculated based on the ALR curve. The resulting value is biased up or down based on the user’s current brightness slider position and then passed on to the Windows graphics stack.
3. The graphics stack receives the computed updated brightness value and forwards it to the new brightness level via WDDM. If the driver exposes a DXGK\_BRIGHTNESS\_INTERFACE\_3 interface, brightness will be passed as nits value. If the driver only exposes a DXGK\_BRIGHTNESS\_INTERFACE\_2 interface, the graphics stack will provide a percentage value.
4. The display driver receives the new brightness level and transitions to the new brightness level.

The following section discusses the various brightness related components that make up an adaptive brightness system.

## Brightness Setting and UI

An on/off setting as well as a brightness slider bar for adaptive brightness is presented in the Settings application under System>Display. In Windows 10 version 1903, users can adjust their current screen brightness preference by using the brightness slider, and also can toggle adaptive brightness on or off.

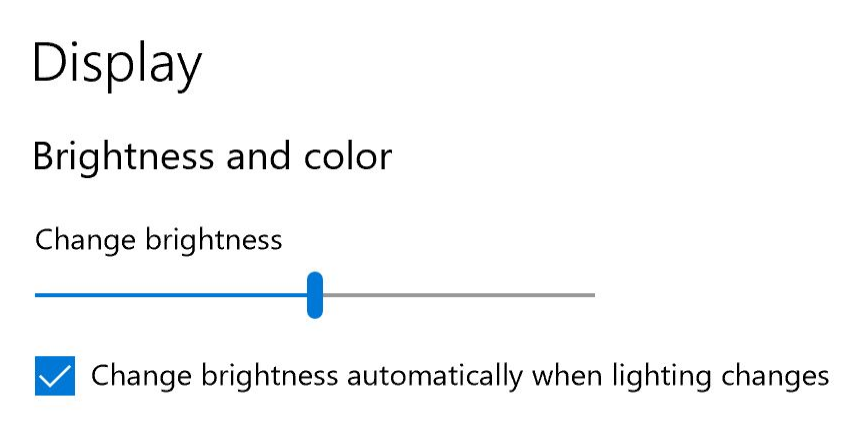


Figure 2: Display settings for adaptive brightness in Settings app

Beginning with Windows 10 Creators Update, percentage based machines (i.e. machines that only expose DXGK\_BRIGHTNESS\_INTERFACE\_2), the brightness slider can behave in two distinctive ways:

* **Legacy behavior**: The slider acts as a simple multiplier taking the ALR computed brightness and multiplying it by a percentage value representing the position of the brightness slider in the Windows Settings UX (Effective brightness = ALR brightness \* slider position). A major limitation of this approach is that users lose control over the brightness in low light condition (as an example of an extreme case, there is no multiplier that can be used to bias an ALR computed brightness of 0% to give a resulting display brightness of 100%). This implementation does not allow the user to scale the screen brightness upwards from this value without disabling automatic brightness functionality and taking manual control.
* **Modern behavior** (recommended for percentage machines): The modern behavior allows users to bias brightness in the most flexible way. Systems can opt-in the newest behavior of the curve by setting the ALRCurveVersion registry key to 2, as mentioned in the [Registry settings](#_ALR_Curve_Version) section

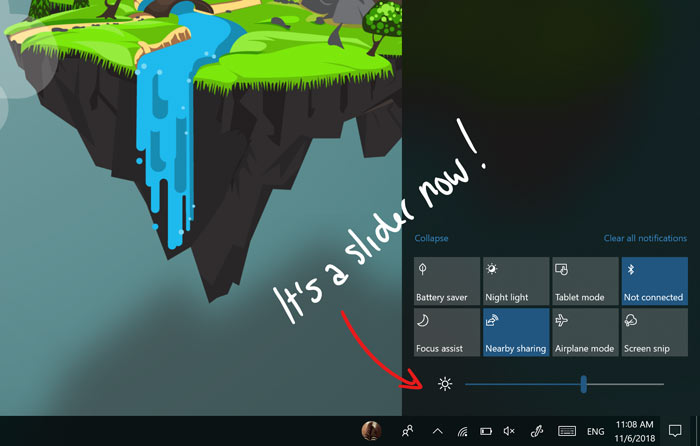
When using the modern behavior, the adaptive brightness algorithm first calculates the optimal output brightness based on the ALR curve defined for the device. This brightness output is then scaled by the brightness slider as follows:

* At the midpoint of the slider – no scaling is applied; the ALR curve is weighted at 100%.
* As the slider increases from 50% to 100% the brightness is scaled proportionally upwards. More weight is given to the slider value and less to the algorithm as it approaches 100% such that at 100% the screen is set to maximum brightness.
* As the slider decrease from 50% to 0% the scaling is reversed, resulting in minimum screen brightness at 0%.

This behavior allows users to easily scale the screen brightness up or down to a desired level while maintaining the benefit of automatic adaptation to the ambient light environment. It also allows for an easy way to override the algorithm to force maximum or minimum screen brightness as needed. This behavior is a fundamental change from the implementation in earlier OS editions.

In Windows 10 version 1809, a third behavior **(nits based brightness behavior)** was added for nits based machines (i.e. machines that expose the DXGK\_BRIGHTNESS\_INTERFACE\_3 interface). The slider always moves as the brightness of the screen changes. On these machines, there is no need to set the ALRCurveVersion registry key. The OS provides an inbox lux to nits mapping curve. When the user changes the brightness, an offset is remembered internally and is used to offset the inbox ALR curve.

Users can also control brightness via the Quick Actions brightness slider (Figure 3).

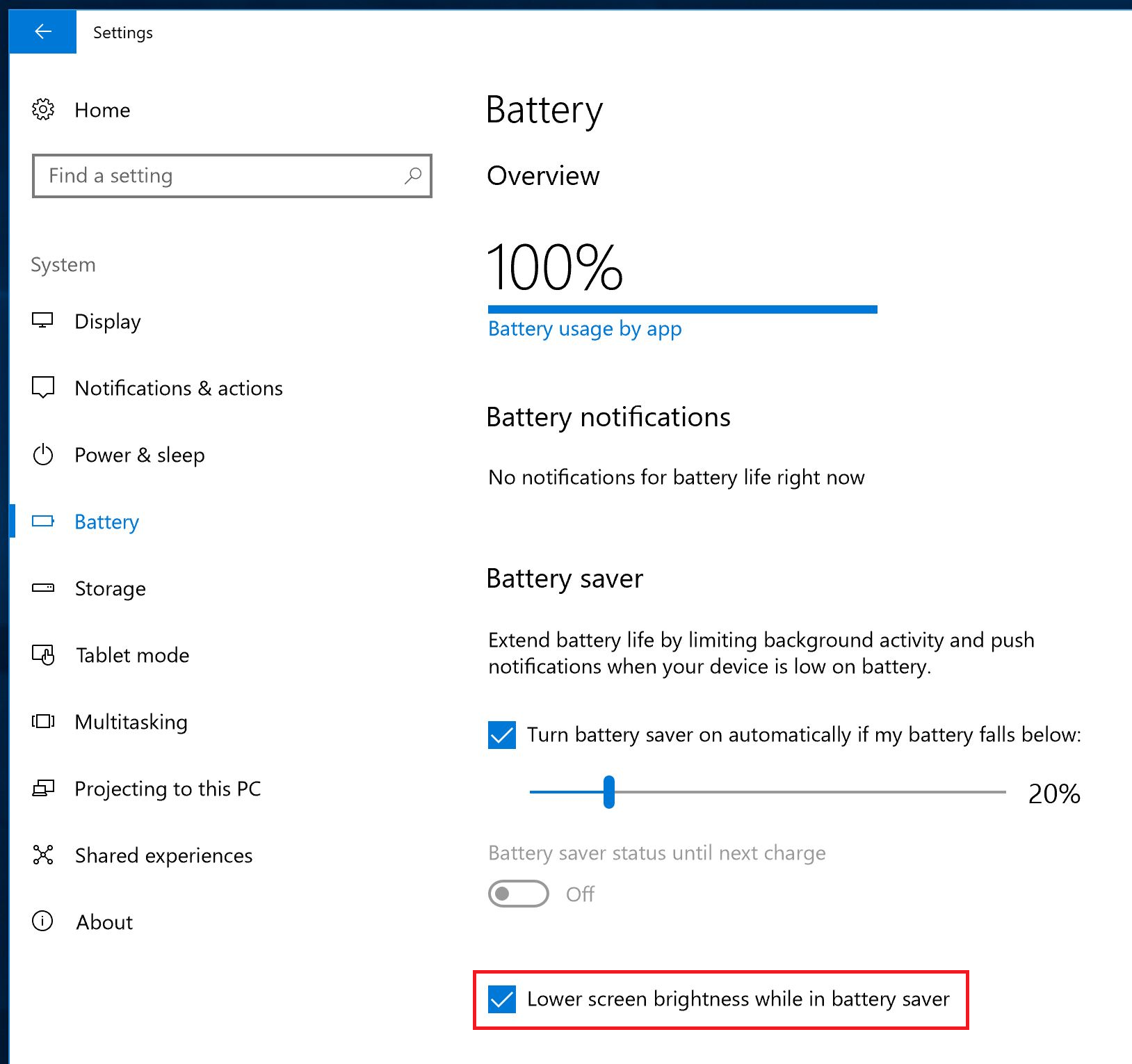


**Figure 3: Quick Actions slider brightness**

### Removal of brightness slider value differences between AC and DC mode

In the past, different brightness slider values were persisted depending on whether the device was plugged in or not (AC vs DC mode). Users often complained about this confusing behavior, so changes were made in Windows 10 version 1903.   
  
The backend for the two different brightness slider values remains the same in Windows 10 version 1903, but beginning in Windows 10 version 1903, when the brightness slider is changed via the OS control panel, OS quick actions, or brightness hotkeys, both backend values get updated to the same value, which fixes the previously confusing UI experience. OEM control panels can still update these backend values independently.

Another toggle under System->Battery in the Settings app allows the system to dim the screen further than it normally would in the current lighting environment when the system is running low on battery power and battery saver mode is enabled. See Figure 4 below for this setting.



**Figure 4: Battery saver mode settings**

## Registry Configuration Parameters

Windows 10 provides a few OEM configurable registry parameters that can be used to fine-tune the behavior of the adaptive brightness feature.

For more information about these parameters, see [Tuning and Configuring Adaptive Brightness Behavior](#_Tuning_and_Configuring) later in this document.

# Light Sensor Features

An ambient light sensor (ALS) is used to measure the lighting conditions of the device’s current environment. When a device is aware of the surrounding lighting conditions, it can use that information to enhance the user experience and save battery life in several ways. Light sensors can support the following types of scenarios:

* Automatic screen brightness
* Automatic Keyboard illumination (Needs additional driver support as this is not supported inbox)
* Light-aware applications

## Automatic Screen Brightness (Adaptive Brightness)

What a user perceives as optimal screen brightness on their device is a function of the current lighting conditions and other factors, such as how the user’s eyes adjust to lighting conditions over time. In low-light conditions, such as in a dark room, the optimal screen brightness is a lower value than in a brightly lit room or daylight outdoor conditions. As surrounding lighting levels increase, the optimal screen brightness needs to increase. In varying lighting conditions, users prefer that the device automatically adjust its own screen brightness. In Windows, this is known as *automatic* or *adaptive brightness*.

Screen display output is measured in **luminance** using units of nits. This is defined as the intensity of light emitted from a surface per unit area in a given direction. The ALS measures **illuminance,** in units of lux, and measures how much light is incident on the device. The adaptive brightness algorithm in Windows takes the illuminance reported by the ALS as input and calculates an appropriate screen brightness output based on OEM customizations, user preferences, and system power considerations.

The adaptive brightness algorithm in Windows operates based on the following principles:

* Consistently present the optimal backlight level in any lighting condition
* Avoid distracting or jarring backlight transitions – especially important in low light environments
* Conserve battery power – do not drive backlight higher than it needs to be for a given lighting condition
* Allow the user to easily and quickly override the algorithm using the brightness slider
* Quickly increase display brightness when moving from a dark to bright environment
* Slow, non-distracting decrease in display brightness when moving to a dark environment
* Hysteresis prevents backlight from adjusting too frequently in a dark environments

Starting from Windows 10 Creators Update, the adaptive brightness algorithm is converged across mobile and desktop devices. Different device types, screens and sensors will need to be independently calibrated to the specific hardware properties and customer use cases, but the underlying algorithm and customization points are converged across mobile and desktop Windows. This document covers how to design and validate an appropriate brightness curve on brightness capable devices.

## Battery Life Considerations

In addition to enhancing the viewing experience, an automatic display brightness control can increase battery life. With display backlight often consuming around 40% of the total system power budget, it is critical that the backlight is never brighter than the user needs. By dimming the display when it is appropriate, a device can reduce power consumption while primarily enhancing user experience and readability automatically in extreme lighting conditions.

On systems that only expose the DXGK\_BRIGHTNESS\_INTERFACE\_2 interface IHVs and OEMs must carefully design the ALR curve. A suboptimal ALR curve will decrease battery life by putting the screen in a brighter than needed state for the given lighting conditions. Please follow the guidance in this document for constructing and calibrating an optimized ALR curve on brightness capable devices. It is critical to the device’s battery life that the ALR curve be tuned to minimize unnecessary power consumption.

Sensor mis-calibration, coarseness of the measured illuminance, and sensor narrow field of view, can significantly limit the system’s battery life in addition to providing a jarring user experience. Special care must be taken when placing and calibrating systems with ambient light sensors.

## Automatic Keyboard Illumination

Traditional keyboards are typically not visible in dark lighting conditions. To resolve this problem, either top illumination or reverse illumination can help users see the keyboard when they type. To optimize power consumption, hardware manufacturers can use an ALS to control the intensity of the illumination based on the lighting conditions. This feature is not supported by default on Windows and needs additional drivers from the device manufacturer to adjust keyboard brightness.

## Light-Aware Applications

To ensure a good user experience across all lighting conditions, Windows automatically adjusts screen brightness (see Automatic Screen Brightness). However, applications can leverage the detected lighting conditions to adapt their behavior and alter their user interface (UI) to maximize contrast and legibility. Combining applications with adaptive brightness can make devices more usable in adverse lighting conditions.

For more information about how to build light-aware applications, see [Implementing Light-Aware UI Using the Windows Sensor and Location Platform](http://download.microsoft.com/download/a/d/f/adf1347d-08dc-41a4-9084-623b1194d4b2/Implementing_Light_Aware_UI.docx).

In Windows, ALS can be accessed by applications via the Windows Runtime API. ALS functionality is exposed by the Windows.Devices.Sensors.LightSensor class. For more information, see the [LightSensor class](https://docs.microsoft.com/en-us/uwp/api/Windows.Devices.Sensors.LightSensor).

In order for proper application functioning, ambient light sensors must be properly calibrated.

# Ambient Light Sensor Integration Checklist

The following high-level checklist is for developers integrating sensor hardware into devices. The rest of this document describes the process and background information in detail.

* Select a suitable backlight source.
* Select a suitable light sensor. (See the [Integrating Light Sensors with Device Hardware](#_Integrating_Light_Sensors) section for additional information)
* Select optimized placement for the light sensor in the device enclosure (see [Optimized Placement for ALS](#_Light_Sensor_Placement)).
* Perform per-model calibration taking into account all factors (coatings, light pipe, sensor configuration, placement, and so on). This should be performed using professional, pre-calibrated light meters (see Ambient Light Sensor Calibration).
* Integrate the sensors on to the device in one of the supported ways.
  + Leverage the inbox HID Sensor class driver (connect the device over USB or I2C HID transport)
* Test the complete device as a light measurement instrument. Use various types of test lighting (incandescent, fluorescent, LED, etc.) at various intensities and compare values reported through the sensor platform with a high-quality industrial/scientific light meter (meter should measure light incident on display of device).
* If the machine supports DXGK\_BRIGHTNESS\_INTERFACE\_2, construct an ambient light response (ALR) curve based on the guidance in the [Tuning and Configuring Adaptive Brightness Behavior](#_Tuning_and_Configuring) section.
* Test the device (and third- party drivers, if pertinent) with the Windows Hardware Lab Kit (HLK) device requirements and related tests. Ensure that it runs correctly and passes all pertinent test cases.
* Ensure that OEM, ODM, and IHV participate in mechanical design reviews for each major revision of device hardware (EV, DV, MP, and so on.). Ensure that sensor implementation is optimized mechanically, optically, and from an electrical engineering perspective.
* Test light sensors and adaptive brightness using the steps mentioned in the [Tests and Validation](#_Tests_and_validation) section.

# Integrating Light Sensors with Device Hardware

Selecting a suitable ALS device is critical. Several things can greatly affect what can be done with the information that the light sensors provide. These considerations include the following:

* The type of sensor (analog versus digital). Digital light sensors are preferred.
* The accuracy, resolution, and field of view of the sensor.
* The dynamic range of the sensor.
* IR and ultraviolet (UV) rejection (human eye response).
* Supported bus technology (digital only).
* The sampling rate (digital only).
* Power consumption.
* Packaging and placement options.

Although these are all important considerations, three factors warrant special consideration:

* Accuracy and resolution: To deliver an optimal user experience for adaptive brightness and light-aware UI in applications, accurate sensor data is required as input. Generally, the more accurate the sensor is, the better the corresponding user experience will be. A good goal for actual ALS calibrated values is a consistent accuracy of within 4 percent of actual lighting conditions.
* Dynamic range: The dynamic range of a light sensor is the ratio between the largest and smallest values that the sensor can report, and defines range of lighting environments in which the sensor can be effective. A low dynamic range light sensor limits the environments in which it can be used. The ambient light sensors mounted on devices designed to be used outdoors (Phone, for example) should support outdoors lighting conditions (typically ranging from 0 lux to 10000 lux or more). The ALS dynamic range for devices designed to be used indoors can be smaller (typically ranging from 0 lux to 1000 lux).
* Granularity: To ensure the best experience the ALS should have a granularity of at most 1 lux when the ambient light is below 25 lux and a granularity of at most 4% of the ambient light when it is above 25 lux. This enables the adaptive brightness algorithm to perform smooth screen brightness transitions.

Table 1. Common lighting conditions and approximate illuminance values

| Lighting condition | Illuminance (lux) |
| --- | --- |
| Pitch black | 1 |
| Very dark | 10 |
| Dark indoors | 50 |
| Dim indoors | 100 |
| Normal indoors | 300 |
| Bright indoors | 700 |
| Dim outdoors (overcast day) | 1,000 |
| Sunlight outdoors | 15,000 |
| Direct sunlight | 100,000 |

## Types of ambient light sensors

Ambient light sensors come in two fundamental types:

* **Analog light sensors** are connected to an embedded controller with an A/D converter and require firmware that can accurately interpret the light sensor data and compensate for various conditions and phenomena that affect readings. Some examples of these phenomena include infrared (IR) light rejection and light frequency compensation. For example, fluorescent lights vary in intensity with the frequency of the AC power that is supplied to the fixture. Analog sensors are typically very inexpensive.
* **Digital light sensors** are more expensive than analog sensors but have advantages over analog light sensors. These sensors can automatically compensate for various conditions and phenomena. Digital light sensors are also extremely compact. Some digital light sensors may provide coarse discreet LUX measurements. The fineness of the readings in low light conditions must be carefully taken into account. Coarse, discreet measurements in low light conditions may result in a jarring brightness experience for the user.

Regardless of which type of light sensor is selected, accurate readings must be taken and exposed to the system.

## Number of Light Sensors

Generally, the more ALSs that are available to measure a lighting condition, the better the estimate of the actual illuminance. As a practical matter, each light sensor adds cost and uses space on the surface of the device.

It is important that manufacturers strive for a solution that provides the system with the most accurate ambient light sensing capability. Inexpensive solutions may rely on a single sensor, but high end hardware can rely on a sensor array to provide the best possible measure. If an OEM chooses to implement multiple ALS sensors (to address issues like a hand or shadow obscuring the ALS), the OEM should expose one logical (consolidated) ALS to the OS and report the most accurate data.

If multiple sensors are exposed to the system, the sensor that should be used for automatic brightness should expose the [DEVPKEY\_SensorData\_LightLevel\_AutoBrightnessPreferred property](https://docs.microsoft.com/en-us/windows-hardware/drivers/sensors/light-sensor-property). Similarly, OEMs may decide to fuse multiple ambient light sensors and expose the result as a virtual light sensor (i.e. a pure software sensor). In this case and if both physical and virtual light sensors are exposed through the sensor device driver interface, the fused sensor must expose the [DEVPKEY\_SensorData\_LightLevel\_AutoBrightnessPreferred property](https://docs.microsoft.com/en-us/windows-hardware/drivers/sensors/light-sensor-property).

## Light Sensor Placement

Correct placement of light sensors is another critical aspect of good system design. The goal of the ALS is to measure the brightness of the environment as perceived by the user. As such, the “best” theoretical sensor location would be between the eyes of the user. As this isn’t practical, the real-world optimum placement of the light sensors is generally on the same plane as the display (facing the user). Sensors placed on the display have the advantage of detecting some glare that may occur on the screen.

Also, avoid placing the light sensor in areas of the computer that are likely to be obscured from the light source or sources by shadows or a user’s hands, fingers, or arm during normal usage. Figure 5 illustrates an example user scenario in which a direct light source is behind the user. A shadow is cast over the lower half of the screen and the base of the computer. This scenario suggests optimal light sensor placement near the top of the screen and facing the user.

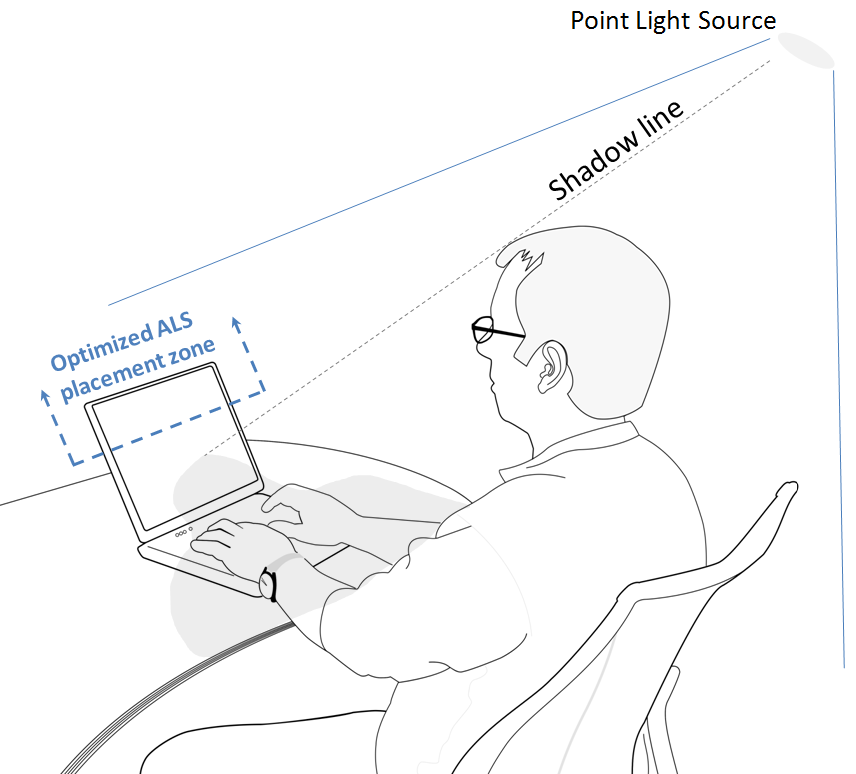


Figure 5. Optimal light sensor placement for a laptop

Additionally, make sure the different configurations a device can take (position of the keyboard in tablet mode vs laptop mode for example) do not block the aperture and do not intersect the sensor’s field of view.

Finally, make sure the sensor’s field of view isn’t intersecting with any noisy source of light (camera flash, keyboard backlight, …) as these can contribute to additional noise or bad readings. Make sure to consider all the different configurations the different configurations a device can take when considering field of view intersecting with noisy sources of light.

## Dealing with invalid light sensor data

Under certain conditions, the ambient light sensor field of view may be obstructed by an object or by the user, leading to the impossibility for the sensor to measure an accurate reading. For example, such condition may happen when the user hand covers the ambient light sensor aperture. Many other cases exist.

The ambient light sensor can indicate this situation to the operating system by sending a new sensor sample with its [PKEY\_SensorData\_IsValid data field](https://docs.microsoft.com/en-us/windows-hardware/drivers/sensors/light-sensor-data-fields) set to FALSE. Proper hardware design should try to minimize the time and scenarios requiring this value to be set to FALSE as such scenario prevents the system from properly controlling brightness. On an ideal system, the ambient light sensor(s) would always be able to measure the ambient light, so this value would always be set to TRUE.

## Light Sensor Filters, Lenses, Enclosures, and Calibration

When engineering a device that includes ALS, the entire system of mechanical, optical, and electrical components related to the ALS needs to be carefully considered and accounted for. The following diagram illustrates the key mechanical components that must be considered and understood when integrating and calibrating ambient light sensor hardware with Windows devices.



Figure 6. Cross section of ALS mechanical components

In this diagram we see the following:

1. Glass (outer surface for screen)
2. Ink coating (black border around screen)
3. Light shielding (used to prevent light bleed/leakage)
4. Light pipe[[1]](#footnote-2) (gathers and directs light to sensor)
5. Light sensor
6. Motherboard

This diagram references two light levels:

1. LUX1: The incident light level for the surroundings of the device (incident on surface of display). This is what is measured and reported by the ALS sensor via the sensor platform.
2. LUX2: The incident light level on the surface of the ALS. **This is not the correct light level to report via the sensor platform as it does not take into account the attenuation[[2]](#footnote-3) factor of the optics[[3]](#footnote-4)**.

**LUX2** should always be lower than **LUX1**

The difference between these two lux values is called the attenuation factor. The attenuation factor accounts for the total percentage of light transmittance[[4]](#footnote-5) between the top surface of the glass (LUX1) and the bare surface of the ALS sensor (LUX2).This is most drastic when painted optics (glass surface) are used. The OEM, with the support of the sensor IHV, should measure the attenuation factor and correct for it in hardware before exposing the lux value to the operating system.

In the below example, let’s assume the total percentage of light transmittance between the top surface of the glass and the bare surface of the ALS sensor is 5%. In order to support the required lux range, the light sensor selected needs to support the following range at the bare sensor:

* Minimum = 1 lux × 0.05 = 0.05 lux
* Maximum = 100,000 lux × 0.05 = 5000 lux

In the firmware or driver (depending on whether a hardware or software ALS solution is being implemented), the following conversion is used to account for the attenuation factor:

Output LUX = LUX1 = LUX2 / (total percentage of light transmittance)

For a bare ALS sensor reading of 100 lux, the following is the resultant output lux:

Output LUX = 100 / 0.05 = 2000 LUX

Note that the entire system should be calibrated with proper light measurement equipment as well. This example only demonstrates the general considerations for part selection and initial calibration prior to formal calibration. Per-unit factory calibration is strongly encouraged for the best and most consistent user experience. Sensors often have accuracy ranges of +/- 20% from unit to unit which can be accounted for via per-unit factory calibration.

Also, an important factor to consider in ALS sensor placement and design is the field of view. The smaller the field of view, the worse the performance of the sensor will be. As a general rule, a 55 degree half angle field of view (110 degrees total) is a fair target. The wider the field of view, the less prone the sensor will be to picking up a single point source of light or an area of shadow that may not accurately reflect the true light environment.

## Sensor Connectivity with HID and SPB

The following diagrams illustrate how to integrate ALS using the HID protocol, and with an IHV-specific driver for SPB.



Figure 7. HID sensor hardware, driver, and software stack

For more information about integrating sensor hardware via the HID protocol, including HID/I2C, please see [HID protocol for sensors](https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/sensor-hid-class-driver).

SPB sensor hardware, driver, and software stack



Figure 8. SPB sensor hardware, driver, and software stack

For more information about integrating sensors via SPB buses, please see the [sensor combo sample driver source code on GitHub](https://github.com/Microsoft/Windows-driver-samples/tree/master/sensors/SensorsComboDriver).

NOTE: the sensor combo driver is designed to host multiple sensors in a single driver, ambient light sensor being one of them. Only the ALS sensor is relevant for the content described in this document, other sensors can optionally be stripped out of the driver.

## Ambient Light Sensor Calibration

### Professional Calibration (Recommended)

Calibration of the ALS in the integrated system using professional, pre-calibrated sensors in a controlled lighting environment is highly recommended. These pre-calibrated sensors, often called "light meters", are available for purchase from electronic equipment vendors and online retailers.

### Other Calibration Techniques

Details on additional ALS monitoring and calibration tools is coming soon. Please look for an update to this document with more information.

# Optimizing Display Brightness Steps and Transitions

The number of brightness levels that a display device exposes is important. Two approaches are possible.

* Controlling brightness using discreet percentage values. Supporting 101 levels (0 through 100) of backlight control is recommended
* Controlling brightness using nits values. Allows for a very fined-grained control of backlight levels, and therefore allows for very smooth and accurate brightness transitions.

The Windows OS is designed to detect the type of brightness interface exposed by the display driver and to select the most appropriate one. If a display driver only exposes the DXGK\_BRIGHTNESS\_INTERFACE\_2 interface, the system will control brightness using percentage values. If a display driver exposes the DXGK\_BRIGHTNESS\_INTERFACE\_3 interface, Windows 1809 will control brightness using nits values. The DXGK\_BRIGHTNESS\_INTERFACE\_3 interface is ignored on down-level Windows versions. If both DXGK\_BRIGHTNESS\_INTERFACE\_2 and DXGK\_BRIGHTNESS\_INTERFACE\_3 interfaces are exposed by the display driver, Windows 1809 will control brightness using nits values while down-level Windows versions will control brightness using percentage values.

## Brightness and display considerations

If the system supports setting brightness in nits by having the display drive expose the [DXGK\_BRIGHTNESS\_INTERFACE\_3](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/dispmprt/nc-dispmprt-dxgk_brightness_set_3) interface, the display should be properly calibrated. Calibration should be performed at various intensities by measuring nits values with a high-quality industrial/scientific nits meter at different locations of the display while displaying a white background. Tools to measure display brightness are called "luminance meters" or "brightness meters", and are available for purchase from electronic equipment vendors and online retailers.

Additionally, the display implementation must be carefully optimized.

Specifically:

* Ensure that display is capable of dimming smoothly across all the accessible brightness levels.
* Make sure that sufficient display brightness levels are exposed to ensure smooth dimming. At least 101 levels are recommended.

## Controlling brightness using nits values

Starting from Windows 1809, the system will control brightness using nits on devices which display driver exposes the DXGK\_BRIGHTNESS\_INTERFACE\_3 interface. Nit (Candela per square meter) is a SI unit of luminance. When a device bears a properly calibrated display and properly calibrated sensor, brightness control should work out of the box. There is no need for any ALR curve on these devices.

For these systems, it is critical that light sensors and display are accurately calibrated. Windows 1809 tolerates small inaccuracies that may occur during the manufacturing process of these systems. Placement of the components, glass transparency, and similar factors can greatly influence the Lux and Nits. Therefore, calibration on nits based brightness systems should be done at least once with the final form factor design as opposed to doing calibration on development form factors and applying the result to the final form factor.

Though not mandatory, per-device calibration during production of each device gives the best end results.

## Controlling brightness using percentage values

Systems that don’t support nits brightness control must support percentage values. On percentage systems, a mapping between the backlight percentages and luminance values is necessary. The mapping of backlight percentages to luminance values should follow an exponential pattern. On nits based brightness systems, since the each nit level is expected to be calibrated, an inbox percentage to nits mapping is provided. This inbox percentage to nits mapping uses human vision and color science research to provide a perceptually linear brightness slider. As long as the nit levels are calibrated correctly as described in [DXGK\_BRIGHTNESS\_INTERFACE\_3](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/dispmprt/nc-dispmprt-dxgk_brightness_set_3), the perceptual luminance difference between 0% and 1% will be automatically equivalent to the perceptual luminance difference between 1% and 2% and so forth.

Human vision is more sensitive to small changes in screen brightness output (nits) at low light levels, thus more backlight levels should be allocated to the lower brightness range to accommodate smoother transitions. The difference between 1% and 2% in nits should be smaller than the difference from 10% and 11% and so on. This means that 50% of the screen’s maximum luminance will not be mapped to the 50% backlight level.

For an acceptable user experience, the lowest level of brightness (i.e. the brightness associated with 0%) must results in a low, but readable display. Users can get into inextricable situations when setting brightness to 0% on devices that map that value to 0 nits as the controls to brighten the screen are not visible anymore. Therefore, the screen must be bright enough at 0% for the user to interact with the UI on the display. On devices supporting the [DXGK\_BRIGHTNESS\_INTERFACE\_3](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/dispmprt/nc-dispmprt-dxgk_brightness_set_3) interface (nits based brightness systems), 0% is automatically clamped to a minimum of 5 nits by the operating system.

The data representing the screen brightness steps is called the *percentage to nits curve*. This curve is separate from, but related, to the *ALR curve* which maps detected ALS brightness, in lux, to brightness percentage. The relationship between ALS input (lux) and screen brightness output (nits) is approximately linear, but given that humans perceive light intensity approximately logarithmically, their respective mappings to brightness percentage are non-linear. See the next sections for more information on developing the ALR curve. The screen brightness percentage to nits curve is generalized by the following formula:

Where,

* *minNits* is a constant and is a reasonable minimum nits value that the screen will output at 0 percent brightness that still allows the user to interact with the display.
* *maxNits* is a constant value representing the maximum nits that the screen supports.
* *percent* is the screen brightness percentage in integral steps from 0 to 100, inclusive.

Figure 9 below illustrates the logarithmic nature of the screen brightness percentage to nits curve for a 435-nit screen with a minimum nits set to 2.

**Figure 9: Theoretical brightness percentage to nits curve for a 435-nit display**

The precise format of the *percentage to nits* curve is display configuration specific and is outside the scope of this document. However, we recommend that all formats represent the curve with as many discrete points as is feasible in order to produce the smoothest possible visual transitions at all brightness levels. The formula given above can be used to calculate the discrete points required for any display configuration. A low number of curve points may result in linear interpolation between points in the *percent to nits* domain, which may be perceptible in the visual domain non-linear and unnecessarily large transitions under low light conditions.

If light sensors control the screen brightness or if the user controls screen brightness with a slider, optimal brightness can be obtained by using all available 101 steps (0 through 100, included). If the user manipulates brightness by using function keys (incrementally raising or lowering the brightness levels), the screen brightness level is stepped up or down automatically in reasonable increments so that the number of required key presses is minimal.

## ACPI considerations

ACPI supports 101 levels of brightness for displays (from 0 to 100 percent in 1‑percent increments), though most platforms only support a limited number of entries and interpolate between them. We recommend exposing the full range of supported entries in the \_BCL method.

# Tuning and Configuring Adaptive Brightness Behavior

Optimal screen brightness data can be collected in several ways. This data can reflect the user’s brightness preference in many different lighting conditions, or it can be calculated and measured based on required power consumption and corresponding battery life. In either case, an optimal user experience should be the result, based on the needs of the intended user.

In the following examples, we base our data and measurements on an optimal user experience on both screen readability and eye comfort.

Screen brightness data varies, depending on the characteristics of the display panel. Depending on the output (maximum brightness) of the display, the user brightness preference (measured as a percentage of maximum brightness) can vary significantly. Consider a portable computer that has a 200-nit (cd/m2 or candela-per-square-meter) display. In a particular lighting condition, the user’s brightness preference may be 80 percent. If the laptop has a 435-nit display, in the same lighting conditions the same user would have a significantly lower brightness preference. When other display characteristics such as reflectivity, resolution, and overall size are considered, it is obvious that optimal screen brightness data must be collected for each type of display panel.

The following example data reflects aggregate user preference for screen brightness under 5 different ambient light conditions. We collected the data by asking users to view an image under controlled lighting conditions where the overall ambient light level was incrementally increased, with time allowed for the users’ eyes to adjust before each illuminance change. Automatic brightness was disabled, the screen brightness was reduced to 0 between each test, and the user was asked to manually select the brightness level most comfortable for viewing under the conditions.

Table 3. Example User Brightness Preference Data

| Ambien light level (Lux) | (Nits) | Slider % |
| --- | --- | --- |
| 2000 | 435 | 100 |
| 1200 | 312 | 86 |
| 400 | 206 | 73 |
| 200 | 164 | 64 |
| 100 | 136 | 58 |
| 40 | 87 | 46 |
| 20 | 44 | 32 |
| 10 | 24 | 24 |
| 0 | 2 | 0 |

**Note:** These example values were chosen for a device with a max output of 435 nits. For devices with a different max nits, these values should be determined using the methodology described.

Figure 10: Nonlinear nature of the illuminance-to-screen brightness data

The trend line shows a mathematical representation of this data. In this case, a logarithmic equation best fits the data. This lets us calculate the optimal screen brightness for any arbitrary lighting condition.

## Lux-to-Nits Relationship

A great screen brightness user experience on a device comes down to the quality of the device’s lux-to-nits response curve: how bright the screen is (in nits) in response to the ambient light measured in the environment (in lux). Practically speaking, in the lux-to-percent and percent-to-nits curves already discussed, the intermediary percentage value is required to map the user control (brightness slider) linearly to the screen brightness output. But the lux-to-nits relationship is what is ultimately perceived by the end user. If the screen output is too high for a given lux, device power is negative impacted and the display may be uncomfortably bright for the user. If it is too low, the user will perceive the screen as too dim.

Thus, it can be valuable to collect user preference data for a given screen in the lux-to-nits format: for a given controlled light environment (lux), what is the average user’s screen brightness preference (nits)? Given a robust dataset from well-designed user studies, this relationship is often found to be consistent across a range of products sharing similar screen characteristics and form-factors.

The following data in Table 4 demonstrates a lux-to-nits dataset generated based on user preference studies for a 435-nit screen, and is plotted in Figure 11.

Table 4. Lux-to-Nits User Brightness Preference Data

| Ambien light level (Lux) | Nits | Slider % |
| --- | --- | --- |
| 2000 | 435 | 100 |
| 1200 | 312 | 86 |
| 400 | 206 | 73 |
| 200 | 164 | 64 |
| 100 | 136 | 58 |
| 40 | 87 | 46 |
| 20 | 44 | 32 |
| 10 | 24 | 24 |
| 0 | 2 | 0 |

**Note:** These example values were chosen for a device with a max output of 435 nits. For devices with a different max nits, these values should be determined using the methodology described.

**Figure 11: Lux-to-nits user preference data plot.**

Armed with a theoretically sound percent-to-nits curve in the display implementation as discussed in the prior Steps and Transitions section, and a robust lux-to-nits user preference dataset similar to this one, the next step is to derive the appropriate ALR curve for the device.

## Ambient Light Response (ALR) Curve

Ambient light response curve is only necessary for systems which display driver expose the DXGK\_BRIGHTNESS\_INTERFACE\_2 interface. Although not recommended, there is a way to override the ALR Curve when using the DXGK\_BRIGHTNESS\_INTERFACE\_3 interface. Additional details are found in the [Tuning and Configuring Adaptive Brightness Behavior](#_Tuning_and_Configuring) section later in this document.

Once the display and ambient light sensor are calibrated, OEMs must focus on creating a brightness curve that works well on their device and form-factor. The brightness curve describes the default response of the algorithm to the ambient light sensor inputs. It is defined as a set of lux to screen brightness percentage mappings stored in the registry which are interpolated into the “curve” by the auto-brightness algorithm.

Guidance on defining brightness curve points:

* OEMs are recommended to define around 10 curve points. The auto-brightness algorithm linearly interpolates the intermediate curve points if the illuminance does not fall on one of the brightness curve points, but the more curve points that are defined, the more finely tuned the brightness response can be to specific lighting conditions.
* The curve must be an increasing function. In other words, each curve point must increment its lux and backlight from its previous curve point.
* When performing user studies to define the ALR curve, the lighting conditions mentioned in the table below should be considered. The list of lux/nit value pairs that are collected during brightness studies can be used to construct brightness curve points. Multiple iterations may be required to refine the lux/nit value pairs. Note that the lux values in the following table are approximate.

| Environment | Approximate lux |
| --- | --- |
| Direct sunlight | 100,000 |
| Full daylight | 15,000 |
| Overcast day | 1,000 |
| Outdoor in vehicle lighting | 900 |
| Sunrise | 2,000 |
| Sunset | 1,500 |
| Office lighting | 250 |
| Dark overcast day | 500 |
| Open space lighting | 300 |
| Office building hallway lighting | 100 |
| Family living room lighting | 100 |
| Morning lighting | 60 |
| Dark room lighting with TV | 40 |
| Dark room lighting | 50 |
| Night sky | 10 |
| Pitch black | 1 |

* Defining a 0 lux curve point associated with a non-zero brightness is strongly recommended. The screen brightness associated with a 0 lux ambient light should allow a user to read and interact with the display’s user interface.

### Understanding Ambient Light Response (ALR) Curve Data

As mentioned in the Optimizing display brightness steps and Transitions section, human vision approximately perceives brightness logarithmically. For example, a 10-nit delta in a 10-lux ambient environment is significantly more perceptible than a 10-nit delta in a 100-lux or 1000-lux environment. This perception difference implies the lux to brightness percentage steps need to follow a roughly logarithmic distribution with relatively more frequent smaller steps clustered towards the low-lux range.

Each curve point in the ALR curve represents a specific mapping between an input from the ALS in lux and the optimum screen brightness percentage. The slope between each point and the adjacent point(s) determines the step size of the lux-to-percent mapping in that range. Take the example in Table 5, and the adjacent lux:percent mappings of 5:15 and 10:22 which implies a 7% change over the 5-lux range or 0.71 lux per step in the 5-10 lux range. Contrast that with the adjacent mappings 700:84 and 1000:90 which computes to a 6% change over a 300 lux range or 50 lux per step. The logarithmic nature of the curve is illustrated in Figure 12.

This sample is for a 435-nit screen and was developed based on the theoretical logarithmic percentage-to-nits curve given in the Steps and Transitions section and the lux-to-nits user data in Lux-to-Nits Relationship section.

The first step is to determine the lux level which will correspond with maximum brightness of the screen. In our example this is taken to be 2000 lux.

Second, select around 10 lux values for your ALR curve points, distributed in the range from 0 to the maximum. More points will lead to smoother transitions and higher fidelity for the curve. In the example we have selected 9 points ranging from 0 to 2000 lux; note that the points are closer together in the low-lux range.

Third, based on the lux-to-nits user preference data, map each one of these lux values to an optimal nits output for the screen. Note that ideally the user study would have gathered data for these precise points; in our example, we have relied on interpolation between points to determine optimal nits.

Finally, take each nits value and find its corresponding percent mapping based on the percentage-to-nits curve in the display implementation. This is the screen brightness percent for each ALR curve point. In our example, using the theoretical percent-to-nits curve, the percent can be calculated mathematically. In a real-world example, this may instead come from a table in ACPI which approximates the curve via linear interpolation.

**Table 5: Sample ALR Curve Points**

|  |  |
| --- | --- |
| Lux | Slider |
| 0 | 0 |
| 10 | 24 |
| 20 | 32 |
| 40 | 46 |
| 100 | 58 |
| 200 | 64 |
| 400 | 73 |
| 1200 | 86 |
| 2000 | 100 |

**Figure 12: Interpolated ALR Curve**

**Please note that this sample curve is for illustrative purposes only.** A high quality device is expected to be implemented with a unique curve that fits the behavioral characteristics of the ALS and display and meets user expectations for the platform. If the display’s percent to nits curve is not implemented efficiently, an ALR curve such as the one above may not perform as expected and may result in wasted power and reduced battery life.

### Constructing a High-Fidelity Ambient Light Response (ALR) Curve from User Preference Data

A high-fidelity ALR curve is critical to ensuring a good user experience and preserving the battery life of the system. A curve that does not produce a bright enough output will frustrate the user and perhaps induce them to disable automatic brightness. On the other hand, a curve that is improperly tuned and results in an overly bright output wastes energy and unnecessarily lowers battery life, also resulting in user frustration. As such, it is critical to construct a well-balanced and high-fidelity ALR curve using input from real users in a wide variety of real-world lighting conditions.

As noted in earlier sections, the ALR curve and the display's brightness percentage to nits mapping are interdependent data sets. A properly performing ALR curve is dependent on a mathematically coherent percentage to nits mapping. If the percentage to nits curve is incorrectly implemented it can be very difficult or impossible to correct for in the ALR curve. Thus the first step to constructing a high-fidelity ALR curve is to ensure that the percentage to nits curve is properly implemented. See the Optimizing and Integrating Display section for more information.

The second step to constructing the ALR curve is to perform user studies to identify optimal brightness levels on your final hardware configuration across a full range of brightness environments as outlined in the ALR Curve section above, ranging from pitch black to direct sunlight. Each one of the listed lighting environments are recommended to be tested, with additional testing around specific lighting environments that may be unique or critical for your device type. For example, if your device is expected to be used more in very bright, very dark, or typical office environments you will want to ensure sufficient fidelity in those ranges to meet user expectations in all conditions.

Consider the inherent brightness of the test imagery in user studies: for example, a typical document or web page with a white background may appear brighter at the same nits level than a photograph or game which may have darker tones. You may want consider a range of possible visual outputs in your study.

Also, keep in mind that brightness perception is dependent on the rate of change of brightness in the users environment. A user who has just transitioned from a bright environment will perceive screen brightness differently than a user who has adjusted to a dim environment and vice versa.

Once you have collected sufficient user data you can aggregate this into a unique ALR curve for the platform. The data should generally conform to a logarithmic curve, though its slope and range will be determined by the performance characteristics of the system, such as the maximum and minimum nits.

# Registry settings

## Registry settings for **Percentage based** brightness

### ALR Curve Setting (ALRPoints)

**Key: (Windows 10 version 1903)**

HKLM\Software\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness\{23B44AF2-78CE-4943-81DF-89817E8D23FD}

**Value:**

|  |  |
| --- | --- |
| ALRPoints (REG\_BINARY) | A binary block where pairs of unsigned 32-bit integers are stored in sequence.  Each pair represents a curve point defining a lux to brightness percentage mapping where Lux is the illuminance value in units of lux with a granularity of 1 lux, and Backlight is percentage units with a 1 percent granularity. |

**Example ALR registry data: (Windows 10 version 1903)**

The following is based on the example ALR curve in Table 4.

[HKLM\Software\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness\{23B44AF2-78CE-4943-81DF-89817E8D23FD}]

"ALRPoints" = hex:00,00,00,00,00,00,00,00,\

0A,00,00,00,18,00,00,00,\

14,00,00,00,20,00,00,00,\

28,00,00,00,2E,00,00,00,\

64,00,00,00,3A,00,00,00,\

C8,00,00,00,40,00,00,00,\

90,01,00,00,49,00,00,00,\

B0,04,00,00,56,00,00,00,\

D0,07,00,00,64,00,00,00

### ALR Curve Version Setting (ALRCurveVersion)

**Key: (Windows 10 version 1903)**

HKLM\Software\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness\{23B44AF2-78CE-4943-81DF-89817E8D23FD}

**Value:**

|  |  |
| --- | --- |
| ALRCurveVersion (REG\_DWORD) | The ALR algorithm version against which the ALR curve data should be interpreted. In order to opt into the Creators Update implementation discussed in this document, the value must be set to 2. |

## Registry settings for **Nits based** brightness

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Default Value** | **Type** | **Value Interpretation** |
| **Autobrightness Lux to Nits ALR curve** | The ambient light response curve that translates lux to nits | Inboxed | REG\_SZ | String (REG\_SZ) with pairs of lux to nits values delimited by ‘,’. Example: “10:15, 100:100, 400:300” would be a curve defined by the points (10 lux, 15 nits), (100 lux, 100 nits), (400 lux, 300 nits). |
| **Stop Transition Conditions** | Stops the current brightness transition when various invalidation factors occur | Bitfield | REG\_DWORD | All bits reserved, except first one is for hands on display |

### Registry location

|  |  |
| --- | --- |
| **Name** | **Registry Location** |
| **Autobrightness Lux to Nits ALR curve** | HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness\{23B44AF2-78CE-4943-81DF-89817E8D23FD}\AutobrightnessLuxToNitsCurve |
| **Stop Transition Conditions** | HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness\{23B44AF2-78CE-4943-81DF-89817E8D23FD}\ShouldStopTransitionDuringHandsOnDisplay |

# Tests and validation

## Validation tools

Sensors can be validated using the SensorExplorer tool that is available from the Windows store. SensorExplorer can be found at <http://aka.ms/sensorexplorer>.

Adaptive brightness integration can be validated using the MonitorBrightnessApp found at <http://aka.ms/sensorstrace>.

The Microsoft Ambient Light Tool (MALT) can be useful to calibrate screen brightness and ambient light sensors. This tool will be required in future HLK requirements. Information about the MALT tool can be found at <http://aka.ms/sensortesting>.

## Display validation

As a first step, you should always run the Graphics and Display [Hardware Lab Kit tests](https://docs.microsoft.com/en-us/windows-hardware/test/hlk/windows-hardware-lab-kit) to validate the display.

Additionally, the following steps ensure an optimal automatic brightness experience:

* Make sure that sufficient display brightness levels are exposed to ensure smooth dimming. At least 101 levels are recommended.
* Ensure that display is capable of dimming smoothly across all the accessible brightness levels.

Test steps:

* 1. Switch the system to manual brightness.
  2. Using a script or a tool, slowly ramp the brightness up and down.
  3. Observe the result on screen. The result should not be jarring to the eye. In a perfectly configured display, it should be impossible to distinguish a transition between any two consecutive brightness levels.
* On a percentage based system, make sure the display is still readable when the lowest brightness level is applied (typically, this is reached when users configure brightness in the Windows settings app as manual brightness and set the brightness slider to 0%).

## Light sensor validation

As a first step, you should always run the Sensors (i.e. Input) [Hardware Lab Kit tests](https://docs.microsoft.com/en-us/windows-hardware/test/hlk/windows-hardware-lab-kit) to validate the ambient light sensor. Ensure that all Windows HLK device and system tests pass.

Additionally, in order to validate proper functioning of the ambient light sensor, please make sure you

* Make sure that the DisplayEnhancementService (or the SensorService, prior to 1809) is started.
* Enable auto-brightness and set the slider to 50%.
* Validate that the display brightness changes when the lighting changes.
* Use a light dimmer to slowly ramp environmental light up and down and ensure that the lux values are **smoothly** ramping up and down. Coarse/discrete light changes will result in sub-optimal screen brightness response and should be avoided.
* Use a professional lux meter and ensure that the ALS readings are accurate. At a minimum verify the following points: 0, 10, 100, 500 and 1000 lux.
* On systems that only customized the ALR curve, test behavior with users to validate ALR data meets user expectations.

## Test cases

| Test Name | ALS Calibration |
| --- | --- |
| **Test Goal** | Ensure the ambient light sensor is properly calibrated for a given set of ambient lights. (I.e. the sensor lux readings are accurate) |
| **Applicable to** | All systems supporting adaptive brightness |
| **Setup/Tools** | A controllable (dimmable) source of light capable of generating different lux levels  A light meter measuring light in lux.  MonitorBrightnessApp (found at <http://aka.ms/sensorstrace> ) to visualize the values reported by the ambient light sensor |
| **Test Procedure** | 1. Set the light meter next to the device. The light meter should be as close to the ambient light sensor as possible without negatively interacting with it.  2. Start the monitor brightness application  3. In a dark room, use the source of light to change the level of ambient light to different lux levels  4. Read the lux meter and the value reported by MonitorBrightnessApp. The values should be identical |
| **Test Variations** | 1. Use different types of controllable lights (such as incandescent lights, CFL, LED). Use different angles |
| **Evaluation (Pass/Fail)** | The LUX values reported by MonitorBrightnessApp should be identical to the values reported by the light meter. |
| **Triage Notes** | Work with your sensor hardware manufacturer to understand how to calibrate the sensor, then calibrate the sensor. |

| Test Name | ALS Granularity |
| --- | --- |
| **Test Goal** | Make sure the ambient light sensor changes in light are fine grained, with no delay |
| **Applicable to** | All systems supporting adaptive brightness |
| **Setup/Tools** | Use a light source with a controllable dimmer to finely ramp the ambient light up and down. The light source should be able to smoothly ramp the light up and down.  Use MonitorBrightnessApp (found at <http://aka.ms/sensorstrace> in the BrightnessTests folder) to visualize the ambient light sensor response. |
| **Test Procedure** | 1. In a dark room, use the dimmer to smoothly ramp the light level up and down  2. Use MonitorBrightnessApp to visualize the ambient light sensor response. The response should match the changes applied to the dimmer. |
| **Evaluation (Pass/Fail)** | The ambient light sensor response should closely match the changes applied to the dimmer. A linear change to the dimmer should result in a linear response from the ambient light sensor (the ALS response should \***not**\* be discreet). A change to the dimmer should be immediately visible in Monitor Brightness App (no delay). |
| **Triage Notes** | Work with your sensor hardware manufacturer to understand how the ambient light sensor transitions can be smoothen out and how delays can be reduced. |

| Test Name | Sleep Transitions |
| --- | --- |
| **Test Goal** | Make sure the ambient light sensor is still functional when coming out of sleep or when the lid is opened |
| **Applicable to** | All systems supporting adaptive brightness |
| **Setup/Tools** | Use a light source to switch the ambient light on and off.  Use MonitorBrightnessApp (found at <http://aka.ms/sensorstrace> in the BrightnessTests folder) to visualize the ambient light sensor response. |
| **Test Procedure** | 1. In a dark room, switch the device on or open the lid.  2. Use MonitorBrightnessApp to visualize the ambient light sensor reading. Make sure the sensor reads a low lux value.  3. Keep MonitorBrightnessApp running throughout the following steps  4. Close the lid or switch the device off  5. Switch the light on, make sure the ambient light is bright  6. Open the lid and/or switch the device on  7. Connect to the desktop and observe the value in MonitorBrightnessApp. The lux value should immediately reflect the actual ambient light. |
| **Evaluation (Pass/Fail)** | The ambient light sensor should send a sample reading when exiting connected standby or when the lid is opened. |
| **Triage Notes** | Work with your sensor hardware manufacturer to understand how the ambient light sensor can be fixed. |

| Test Name | %-to-NITs Curve |
| --- | --- |
| **Test Goal** | Validate the percentage-to-nits curve in the display driver |
| **Applicable to** | Systems supporting percentage brightness |
| **Setup/Tools** | A brightness (nits) meter  Optional: Use the DimToBright.ps1 script (found at <http://aka.ms/sensorstrace> in the BrightnessTests folder) to linearly ramp the brightness up and down. |
| **Test Procedure** | 1. Make sure the screen is displaying as much white as possible (one option: open notepad and maximize it on the screen)  2. Set brightness control to manual brightness.  3. For each position of the slider, measure the nits value for a white portion of the screen. The nits values should follow a logarithmic curve to be perceived by the eye as linear. |
| **Evaluation (Pass/Fail)** | The nits values should follow a logarithmic curve to be perceived by the eye as linear. Please note that the script used in the following step can help make sure the curve is perceived by the eye as linear. |
| **Triage Notes** | Work with your graphics/display hardware manufacturer to understand how configure the percentage-to-nits. |

| Test Name | Smooth Transitions |
| --- | --- |
| **Test Goal** | Make sure the display panel smoothly changes brightness |
| **Applicable to** | All systems supporting adaptive brightness |
| **Setup/Tools** | Use the BrightToDim.ps1 script (found at <http://aka.ms/sensorstrace> in the BrightnessTests folder) to linearly ramp the brightness up and down. You will want to switch your device to manual brightness. The script will ramp the screen brightness down from 100 to 0.  Optional: use a brightness (nits) meter to measure the screen brightness |
| **Test Procedure** | 1. Make sure the screen is displaying as much white as possible (one option: open notepad and maximize it on the screen)  2. Start the BrightToDim.ps1 script and observe the screen behavior. The screen should transition as smoothly as possible, without any jumps in brightness. |
| **Evaluation (Pass/Fail)** | The brightness on screen should smoothly transition up and down, there should \***not**\* be any visible jumps in brightness. |
| **Triage Notes** | Work with your graphics/display hardware manufacturer to understand how to smoothen the display panel brightness changes. |

| Test Name | 0% Brightness |
| --- | --- |
| **Test Goal** | Make sure 0% brightness does \***not**\* turn the display off and keeps the Windows UX visible to the user |
| **Applicable to** | Systems supporting percentage brightness |
| **Setup/Tools** | Optional: use a brightness (nits) meter to measure the screen brightness |
| **Test Procedure** | 1. Switch the machine to using manual brightness  2. In the system settings UX, drag the slider to the minimum setting.  3. Observe the screen, the display should be dim but should still be readable. The display should \***not**\* be turned off. |
| **Evaluation (Pass/Fail)** | 0% brightness should \***not**\* turn the display completely off. In the lowest brightness setting, users should be able to find their way through the Windows UX to bring the brightness back up to a comfortable level |
| **Triage Notes** | Work with your graphics/display hardware manufacturer to understand how to change the lowest brightness value. |

| Test Name | V2 ALR Curve |
| --- | --- |
| **Test Goal** | Make sure the system is using a V2 ALR curve. This test is only applicable to percentage brightness systems. |
| **Applicable to** | Systems supporting percentage brightness |
| **Setup/Tools** | Use regedit.exe to make sure the ALRCurveVersion value is set to 2  The DISM tool to configure the OS image |
| **Test Procedure** | Use DISM to configure your OS image so that the ALRCurveVersion is set to 2:  [HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness\{23B44AF2-78CE-4943-81DF-89817E8D23FD}]  @="Adaptive Brightness Feature"  "ALRCurveVersion"=dword:00000002 |
| **Evaluation (Pass/Fail)** | Ensure the registry key is present in the registry.  Ensure the registry key should be set to 2 |
| **Triage Notes** | Make sure the OS image you created has this registry key set. You can use the DISM tool to configure this registry key in your OS image. |

| Test Name | ALR Curve Configuration |
| --- | --- |
| **Test Goal** | Configure the ALR curve brightness |
| **Applicable to** | Systems supporting percentage brightness |
| **Setup/Tools** | A light meter  A brightness (Nits) meter  Regedit.exe to configure the ALR curve  The DISM tool to configure the OS image |
| **Test Procedure** | 1. As a starting point, you can use the theoretical curve suggested in the documentation.  2. Tweak the curve based on user studies. User studies may involve asking a bunch of users for their brightness preferences for different ambient light values  3. Once the curve is defined, use DISM to configure your OS image with your ALR curve stored in the registry under  [HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AdaptiveDisplayBrightness\{23B44AF2-78CE-4943-81DF-89817E8D23FD}]  @="Adaptive Brightness Feature"  "ALRCurveVersion"=dword:00000002 |
| **Evaluation (Pass/Fail)** | Ensure the curve is properly stored in the registry. |
| **Triage Notes** | Make sure the OS image you created has the right registry key set. You can use DISM to configure this registry key in your OS image. |

| Test Name | Starting and Stopping Ambient light sensor |
| --- | --- |
| **Test Goal** | Make sure the ambient light sensor is reporting the correct value when starting/stopping the sensor |
| **Applicable to** | All systems supporting adaptive brightness |
| **Setup/Tools** | Use SensorInfo (Download from Windows store) to visualize the ambient light sensor reading. |
| **Test Procedure** | 1. Open SensorInfo. Switch to the light sensor tab. Keep the app running and snap the app to the left side of the screen. Keep it visible throughout the steps.  2. Open the Display settings page and snap it to the right side of the screen.  3. Toggle on "Change brightness automatically when lighting changes" checkbox.  4. Repeatedly toggle off and on "Change brightness automatically when lighting changes" checkbox to simulate starting and stopping sensors.  5. Verify that the lux value keeps reporting the same value/correct ambient light value. |
| **Evaluation (Pass/Fail)** | Verify that the lux value keeps reporting the correct ambient light reading when starting/stopping sensors. |
| **Triage Notes** | Work with your sensor IHV to fix any issue if the lux value fluctuates when starting/stopping the ambient light sensor. |

| Test Name | Reactivity to brightness changes |
| --- | --- |
| **Test Goal** | Make sure the backlight panel is reactive to brightness changes |
| **Applicable to** | All systems supporting adaptive brightness |
| **Setup/Tools** | ReactivityScript.ps1 script (found at <http://aka.ms/sensorstrace> in the BrightnessTests folder) to switch the brightness up and down. You will want to switch your device to manual brightness. The script will switch the screen brightness up immediately between 0 and 100.  Optional: use a brightness (nits) meter to measure the screen brightness |
| **Test Procedure** | 1. Make sure the screen is displaying as much white as possible (one option: open notepad and maximize it on the screen)  2. Start the ReactivityScript.ps1 script and observe the screen behavior. The screen should transition as quickly as possible, without any delays in brightness. |
| **Evaluation (Pass/Fail)** | The brightness on screen should transition up and down every 2 seconds, there should \***not**\* be any visible latency (i.e. max latency = 200ms) between the time the script changes brightness and the time the brightness change is visible on the screen. |
| **Triage Notes** | Work with your graphics/display hardware manufacturer to understand how to smoothen the display panel brightness changes. |

| Test Name | Battery Saver Mode |
| --- | --- |
| **Test Goal** | Make sure the brightness changes correctly when battery saver mode turns on and off. |
| **Applicable to** | All systems supporting brightness. |
| **Setup/Tools** | Update to the latest version of Windows 10. |
| **Test Procedure** | 1. Set brightness to any arbitrary value 2. Under Battery Saver Settings: tick “Lower screen brightness while in battery saver” 3. Toggle “Battery saver status until next charge” on. 4. Verify that brightness is dimmer. 5. Toggle “Battery saver status un next charge” off. 6. Verify that brightness is brighter. |
| **Evaluation (Pass/Fail)** | Steps 4 and 6 should be true for a pass. |
| **Triage Notes** | Work with Microsoft if this is not occurring as expected. |

# Conclusion

Integrating ALS sensors into your Windows device can greatly improve user experience. Light sensors in Windows 10 version 1903 provide many opportunities for OEMs and hardware manufacturers. Adaptive brightness and light-aware applications represent just two examples. Correctly selecting, integrating and calibrating light sensor hardware helps take advantage of drivers and features that are now included in Windows.

# Additional documentation

## Sensor Driver Requirements

* [Sensor device driver reference for Windows 10](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/index)
* [Overview of universal sensor driver model](https://docs.microsoft.com/en-us/windows-hardware/drivers/sensors/overview-of-converged-sensor-driver-model)

# Resources

Public documentation and resources:

* [Sensor device driver reference for Windows 10](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/index)
* [Windows HLK (Windows Hardware Lab Kit)](https://docs.microsoft.com/en-us/windows-hardware/test/hlk/windows-hardware-lab-kit)
* [Overview of universal sensor driver model](https://docs.microsoft.com/en-us/windows-hardware/drivers/sensors/overview-of-converged-sensor-driver-model)
* [Windows sensors driver samples](https://github.com/Microsoft/Windows-driver-samples/tree/master/sensors)
* [HID protocol for sensors](https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/sensor-hid-class-driver)

* [Windows Sensor Calibration and Configuration Persistence](http://msdn.microsoft.com/en-us/library/windows/hardware/jj159304.aspx)

1. Note that light pipes are usually not needed, and in many cases may degrade ALS performance. Please consult with the light sensor manufacturer for guidance regarding these kinds of optical components. [↑](#footnote-ref-2)
2. Attenuation factor corresponds to how much light is blocked by the various components between the outer surface of the device (typically glass) and the sensing surface of the ALS. Attenuation can be calculated as follows: A = (1 - transmittance) [↑](#footnote-ref-3)
3. The ALS sensor reports the ambient light intensity that it senses. Due to the transmissibility (attenuation) of the optics, the raw ALS readings report incorrect (attenuated) Lux values and should not be used without correction. The transmissibility is the characteristics of the optics that reduce ambient light intensity and also reject infrared (IR) light. If the optics are painted with ink for visible appearance, an attenuation factor must be used to compensate for the corresponding attenuation. [↑](#footnote-ref-4)
4. Transmittance is the ratio of the light level at the surface of the ALS divided by the ambient light level surrounding the device. [↑](#footnote-ref-5)