Implementing Light-Aware UI by Using the Windows Sensor and Location Platform

October 27, 2008

Abstract

This paper covers the use of ambient light sensor data, and how user interface features and program content can be optimized for many lighting conditions.

This information in this paper applies to the Windows® 7 operating system.

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

For the latest information, see:   
 <http://www.microsoft.com/whdc/>

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Windows, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

s mentioned herein may be the trademarks of their respective owners.

Revision History

|  |  |
| --- | --- |
| **Date** | **Change** |
| 10/24/2008 | PDC 2008 |

Contents

[Introduction 4](#_Toc212889160)

[Ambient Light Sensors 4](#_Toc212889161)

[Scenario: Using Your Laptop to Navigate to a Restaurant 4](#_Toc212889162)

[Light-Aware UI Fundamentals 6](#_Toc212889163)

[Scale 6](#_Toc212889164)

[Varying Font Size 7](#_Toc212889165)

[Zooming Content 7](#_Toc212889166)

[Altering Vector Graphic Rendering Properties 7](#_Toc212889167)

[Contrast 8](#_Toc212889168)

[Color 8](#_Toc212889169)

[Examples of Light-Aware UI 8](#_Toc212889170)

[Optimizing the User Experience 10](#_Toc212889171)

[Introducing the Windows Sensor and Location Platform 10](#_Toc212889172)

[Windows Sensor and Location Platform Summary 10](#_Toc212889173)

[The Sensor Device Driver Interface (DDI) 10](#_Toc212889174)

[The Sensor Application Programming Interface (API) 10](#_Toc212889175)

[Permissions and Configuration UI 11](#_Toc212889176)

[Developer Tools and Resources 11](#_Toc212889177)

[Understanding and Interpreting LUX Values 11](#_Toc212889178)

[Interacting with Sensors through the Sensor API 13](#_Toc212889179)

[Getting Your Project Started with the Sensor API 13](#_Toc212889180)

[Using ISensorManager and ISensorManagerEvents 13](#_Toc212889181)

[Accessing Sensors 13](#_Toc212889182)

[Requesting Sensor Permissions 14](#_Toc212889183)

[Sensor Event Handling 15](#_Toc212889184)

[Using ISensor and ISensorEvents 15](#_Toc212889185)

[Sensor Properties 15](#_Toc212889186)

[Subscribing to Events 17](#_Toc212889187)

[Handling Events and Interpreting Ambient Light Sensor Data 18](#_Toc212889188)

[Light Sensor Data Types 19](#_Toc212889189)

[Illuminance 19](#_Toc212889190)

[Color Temperature 19](#_Toc212889191)

[Chromacity (also known as Chromaticity) 19](#_Toc212889192)

[Handling Data from Multiple Light Sensors 19](#_Toc212889193)

[Conclusion 20](#_Toc212889194)

[Call to Action 20](#_Toc212889195)

[For More Information 20](#_Toc212889196)

[Appendix A 21](#_Toc212889197)

[Appendix B 22](#_Toc212889198)

[Sensor API Concepts 22](#_Toc212889199)

[Sensor Categories 22](#_Toc212889200)

[Sensor Types 22](#_Toc212889201)

[Sensor Data Types 22](#_Toc212889202)

# Introduction

Computers today are more mobile than ever. From small laptops to Tablet PCs, many computers can go wherever the user wants to go. However, if you have tried to use a laptop in the car or outside in direct sunlight you have probably discovered that the computer is often not usable because of constraints like screen readability.

So what if computers could adapt to their surroundings and provide an optimized experience based on environmental conditions and other factors? Would your laptop be more useful to you if you could use it in the car, next to a window, or outdoors? The Windows® 7 Sensor and Location platform enables the computer to be aware of, and adaptive to, its surroundings.

This paper covers the use of ambient light sensor data, and how user interface features and program content can be optimized for many lighting conditions.

# Ambient Light Sensors

Ambient light sensors expose data that can be used to determine various aspects of the lighting conditions present where the sensor is located. Ambient light sensors can expose the overall brightness of an environment (Illuminance) and other aspects of the surrounding light, such as chromaticity or color temperature.

Computers can be more useful in several ways when the system is responsive to lighting conditions. These include controlling the brightness of the display (a new, fully supported in-box feature for Windows 7), automatically adjusting the lighting level of illuminated keyboards, and even brightness control for other lights (such as button illumination, activity lights, and so on).

End-user programs can also benefit from light sensors. Programs can apply a theme that is appropriate for a particular lighting condition, such as a specific outdoor theme and indoor theme. Perhaps the most important aspect of light sensor integration with programs is readability and legibility optimizations that are based on lighting conditions.

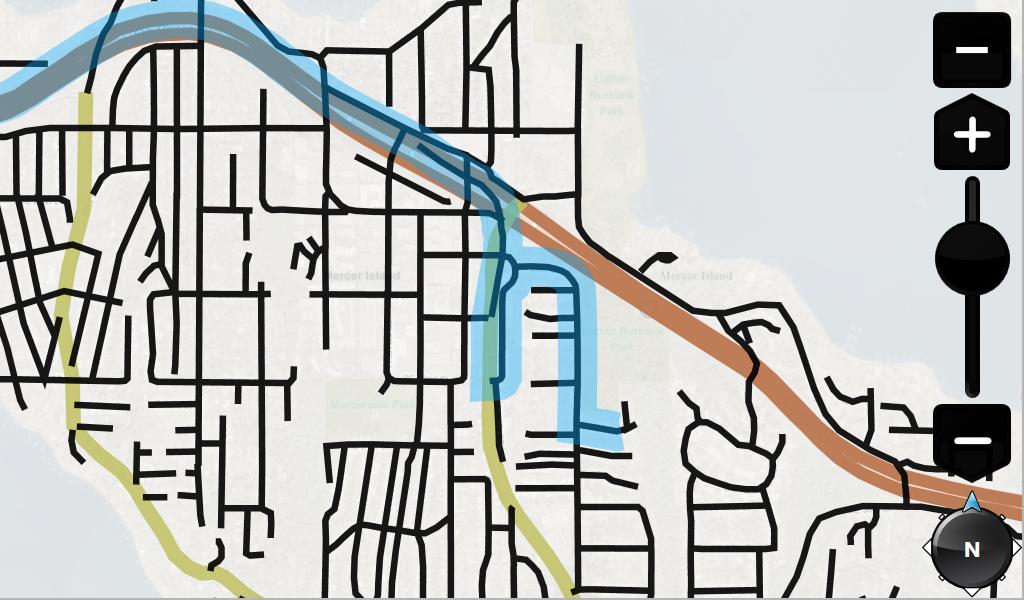
## Scenario: Using Your Laptop to Navigate to a Restaurant

Suppose you want to use your computer to help you navigate to a new restaurant. You start out in your house looking up the address of the restaurant and planning your route.

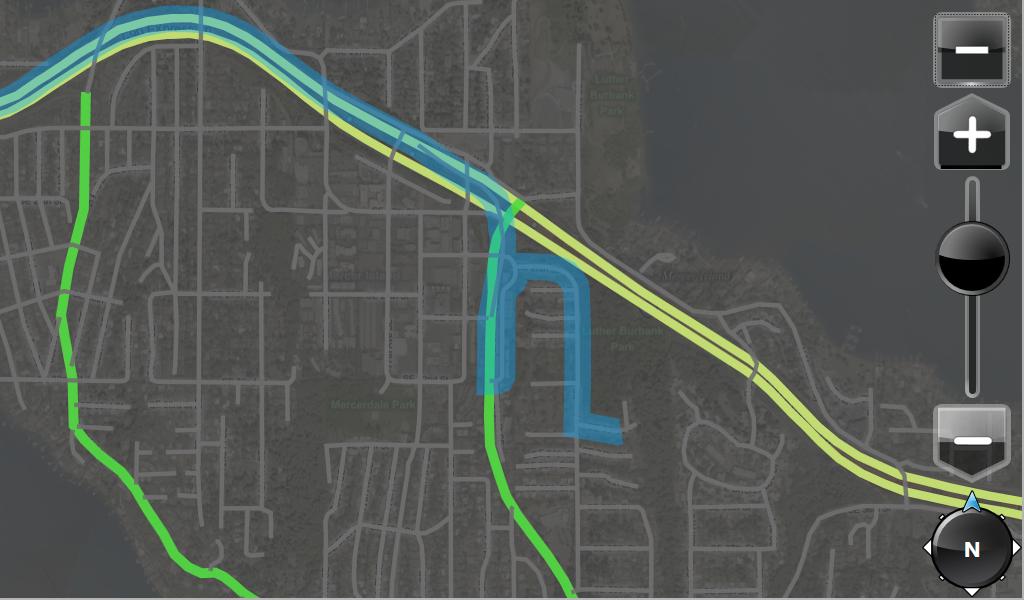
The following screenshot shows how your navigation program could optimize its UI for indoor lighting conditions and show detailed information.



When you go outside to your car, you encounter direct sunlight, which makes the laptop’s screen difficult to read. The following screenshot shows how your program could alter its UI to maximize legibility/readability in direct light. In this view, much of the detail has been omitted and contrast is maximized.



As you get closer to the restaurant, evening approaches and it gets dark outside. In the following screenshot, the UI for the navigation program has been optimized for low-light viewing. By using darker colors overall, this UI is easy to glance at in the dark car.



In the remainder of this paper, we’ll look at what programs can do to optimize for various lighting conditions and how the Windows 7 Sensor and Location platform can be used to detect lighting conditions to enable light-aware UI.

# Light-Aware UI Fundamentals

The term “light-aware UI” refers to a program that uses light sensor data to optimize its content, controls, and other graphics for an optimum user experience in many lighting conditions, ranging from darkness to direct sunlight. Perhaps the most important optimizations are legibility, readability, and interactions in direct sunlight because screens do not typically perform well in these conditions. In this section, we focus on three UI metrics (scale, color, and contrast) that can be changed to optimize the visual user experience.

## Scale

In general, larger objects are easier to see. When the computer is in adverse lighting conditions (such as in direct sunlight) making content larger can help to improve the legibility and interactiveness of that content.

The following images show a laptop in direct sunlight with typical screen brightness and zoom levels (on the left) and a laptop in the same lighting conditions with light-aware UI (on the right):

|  |  |
| --- | --- |
| 40% brightness, normal zoom level | 100% brightness, increased zoom level |
| |  | | --- | |  | |  |

The following are examples of how scaling content can be implemented in your program.

### Varying Font Size

If you increase the size of the font that is used to display text, the text is more legible in adverse lighting conditions. Font style, font face, and other characteristics can also be varied to optimize legibility and readability. For example, sans serif fonts are typically easier to read than serif fonts:

| Sans serif font | Serif font |
| --- | --- |
| Example text (Verdana, 11pt.) | Example text  (Times New Roman, 11 pt.) |

### Zooming Content

If your program implements zooming functionality, zooming can be used to scale the content. Zooming in enhances legibility while zooming out allows the program to display more content.

### Altering Vector Graphic Rendering Properties

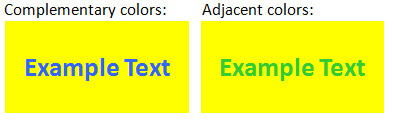
If your program has vector graphic primitives that are rendered (such as lines, circles, and so on), the characteristics of the rendering can be altered to optimize legibility. For example, if your program renders rectangles, the width of the lines that are used to render the rectangles could be scaled (wider for outdoors and narrower for indoors) to optimize the appearance and legibility of the vector graphic content.

## Contrast

When LCD screens are used in bright lighting conditions, the overall contrast of the screen is reduced. By flooding the screen with light (from the sun, for example), the user’s perception of dark areas on the screen is reduced. In general, this makes increasing content and UI contrast important when in high-ambient lighting conditions. It may be desirable to use a monochrome content scheme to maximize contrast in these lighting conditions. Another way to increase contrast is to replace low-contrast content (such as an aerial photo mode in a mapping program) with high-contrast elements (such as black-on-white street vector graphics mode).

## Color

The colors that a program uses to display its content can have a drastic affect on the overall user experience and legibility of rendered content. By changing color contrast based on ambient light, you can make content more readable in adverse lighting conditions, such as bright outdoor light or dark interior light.

One way to increase color contrast is through color saturation. Another way is by using complementary colors instead of adjacent colors for better readability. Complementary colors are pairs of colors that are of opposite hue, such as blue and yellow. The following is a side-by-side example of how using complementary colors can help improve color contrast:

## Examples of Light-Aware UI

Now that some basic principles for optimizing your UI for different lighting conditions have been outlined, let’s take a look at how this makes a difference when viewing content outdoors in direct sunlight. The following images are side-by-side comparisons of laptops in direct sunlight.

The first example is of an RSS reader on a laptop. One laptop has light-aware UI and the other does not.

|  |  |
| --- | --- |
| UI with light-awareness, 100% screen brightness | UI without light-awareness, 40% screen brightness |
| UI_Light_Aware.jpg | UI_Not_Light_Aware.jpg |

The picture on the right shows what content typically looks like outdoors when a laptop is using its battery usage display settings. The picture on the left shows the combination of “Adaptive Brightness” and light-aware UI, and how these features can work together to increase screen readability.

The following example is of a navigation program as seen outdoors with light-awareness turned on and turned off.

|  |  |
| --- | --- |
| Navigation UI, with light-awareness | Navigation UI, without light-awareness |
| Navigation_Aware.png | Navigation_Not_Aware.png |

These images correspond to the screen shots shown earlier in this paper, but this time as viewed in outdoor lighting conditions with the same screen brightness. The “indoor” content is not usable outdoors, whereas the “outdoor” content is easily legible. Also note how the reflections are minimized when a black-on-white background scheme is used (right photo).

## Optimizing the User Experience

When implementing light-aware UI, the user’s reaction to the program’s behavior should be carefully considered. Specifically, it is best to avoid jarring transitions or frequent changes to the program’s content and UI. Smooth and gradual transitions that take place only as needed are best. Ideally, your program should be tested in real-world lighting conditions and user scenarios with users. Finally, it might be advantageous to expose a mechanism that allows users to manually change the program’s light optimizations or disable the functionality.

# Introducing the Windows Sensor and Location Platform

We have looked at some interesting scenarios around light awareness in programs. You may be wondering why these scenarios are not already part of the mainstream for today’s computers. One of the main reasons for the lack of adoption of sensor technology on computers is a lack of standards and developer resources for dealing with sensor devices on Windows.

This problem is addressed in Windows 7 by a comprehensive new platform for sensor and location devices. Windows 7 provides a common driver model, API, permissions model, and configuration UI for interacting with sensor and location devices. This platform is called the Windows Sensor and Location platform.

## Windows Sensor and Location Platform Summary

The following components make up this new platform for sensor and location devices.

### **The Sensor Device Driver Interface (DDI)**

The sensor DDI is a framework for writing sensor drivers for Windows 7. This DDI is based on user-mode driver framework (UMDF). Sensor drivers use a driver utility library called the Sensor Class Extension. This sensor class extension implements functionality including:

* Common functionality that would otherwise be duplicated driver code.
* Exposing sensors to the sensor platform.
* Standard interfaces and for incoming and outgoing calls to, and from, the device driver.
* Permissions enforcement between sensor devices and programs.

### **The Sensor Application Programming Interface (API)**

The Sensor API provides a common way of discovering and interacting with sensors on Windows including:

* Definitions for sensor interactions.
* A set of standard interfaces for discovering sensors.
* A set of standard interfaces for interacting with individual sensors.
* A mechanism for requesting permissions from the user.
* A mechanism for creating a virtual device node for logical sensors.

### **Permissions and Configuration UI**

Sensors can expose sensitive data, such as a user’s current location. To safeguard the user from unwanted disclosure of private data, the Windows 7 Sensor and Location platform has mechanisms that allow a user to control how sensor data is exposed.

Windows 7 now includes the Location and Other Sensors Control Panel. If you have sensor devices are installed on your computer, you can configure permissions (either global on/off, or per device/per user), view sensor metadata, and change the description for sensor devices. Also included is UI that appears to the user when API calls are made from programs requesting permissions.

### **Developer Tools and Resources**

The Windows 7 Software Development Kit (SDK) and WDK feature sensor platform documentation, tools, and samples that make it easy for driver or software developers to write sensor drivers and sensor-enabled programs.

Another resource that can be used to develop sensor-enabled programs is the Windows 7 Sensor Development Kit, which includes a sensor development board, sample firmware, sample driver code, and sample programs with source code. For more information about the Windows 7 Sensor Development Kit, see the link to the official Web site for sensor devices at the end of this document.

# Understanding and Interpreting LUX Values

The primary sensor data type for ambient light sensors is illuminance in lux (lumens per square meter). The principles outlined in this paper are all based on taking lux values as input and reacting to that data in a program.

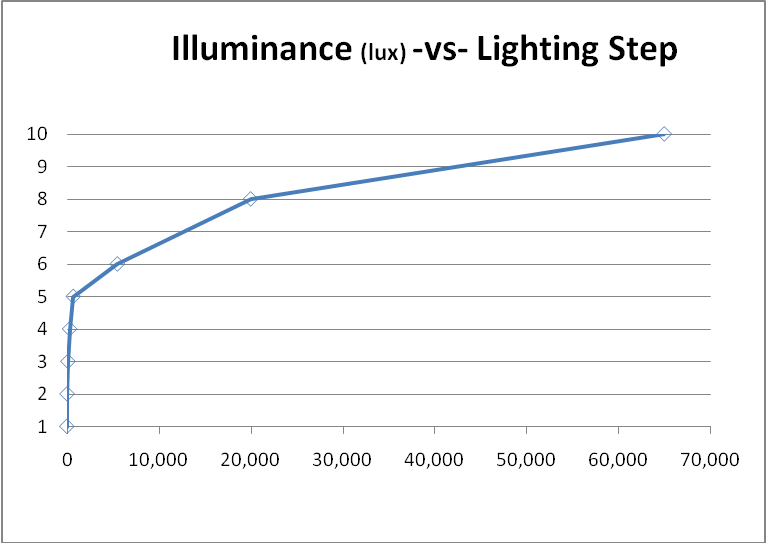
Lux readings are directly proportional to the energy per square meter that is absorbed per second. Human perception of light levels is not so straightforward. Human perception of light is complicated because our eyes are constantly adjusting and other biological processes are affecting our perception. However, we can think of this perception from a simplified perspective by creating several “ranges of interest” with known upper and lower thresholds.

The following example data set represents rough thresholds for common lighting conditions, and the corresponding lighting step. Here, each lighting step represents a change in lighting environment.

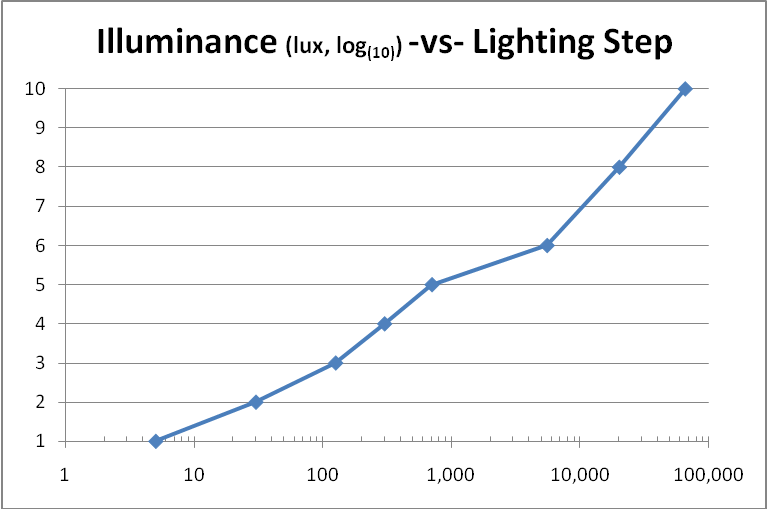
Note This data set is for illustration and may not be completely accurate for all users or situations.

| **Lighting condition** | **From (lux)** | **To (lux)** | **Mean value (lux)** | **Lighting step** |
| --- | --- | --- | --- | --- |
| Pitch Black | 0 | 10 | 5 | 1 |
| Very Dark | 10 | 50 | 30 | 2 |
| Dark Indoors | 50 | 200 | 125 | 3 |
| Dim Indoors | 200 | 400 | 300 | 4 |
| Normal Indoors | 400 | 1,000 | 700 | 5 |
| Bright Indoors | 1,000 | 5,000 | 3,000 | 6 |
| Dim Outdoors | 5,000 | 10,000 | 7,500 | 7 |
| Cloudy Outdoors | 10,000 | 30,000 | 20,000 | 8 |
| Direct Sunlight | 30,000 | 100,000 | 65,000 | 9 |

If we visualize this data by using the mean values from this table, we see that the “lux-to-lighting step” relationship is nonlinear:



However, if we view this data with a logarithmic scale (log(10)) on the x-axis, we see a roughly linear relationship emerge:



**Example Transform**

Based on the sample data set for ambient light sensors discussed previously, we could arrive at the following equation to map our lux values to human perception. In this example we’ll expect a range of 0 lux to 1,000,000 lux:

**Lightnormalized =** Log10(x) / 5.0

This equation gives us values that vary roughly linearly between 0.0 and 1.0. This indicates how human-perceived lighting changed based on the example data set that was shown previously.

Now that we understand how to interpret the lux data coming from the ambient light sensors and based on the specific data sets we have looked at, there are two recommended ways of interpreting and using the data:

* Optimal implementation

Apply a transform to the data so that the “normalized light level” can be used in direct proportionality to program behaviors or interactions. An example of how this might be used would be to vary the size of a button in your program, where the size of the button is directly proportional to the normalized data (or a range of the normalized data, corresponding to outdoors, for example).

* If smooth transitions are not feasible.

Deal with ranges of lux data, and map program behaviors and reactions to the upper and lower thresholds of these ranges of lux data. This is a simple way to respond to lighting conditions and may not yield the optimal user experience.

# Interacting with Sensors through the Sensor API

If you have never used the Sensor API, see Appendix B in this document for information about sensor categories, types, data types, and so on.

## Getting Your Project Started with the Sensor API

To start working with the Windows Sensor and Location platform, you must install the Windows 7 SDK. This installs the Windows 7 SDK documentation, tools, and samples. For more information, see Appendix A in this document.

## Using ISensorManager and ISensorManagerEvents

The ISensorManager object is the root co-creatable object for the Sensor API. It is the interface by which you discover and connect to sensors. It is also used for requesting permissions.

### Accessing Sensors

The first step is to discover and connect to the sensor. Sensors can be obtained by category, type, or unique ID (unique to each sensor that is enumerated on a system).

The following example code shows how to create an ISensorManager object and how to find and connect to an ambient light sensor on the system.

Note For clarity, error-handling code is omitted from all example code in this paper. You should check all HRESULT values that are returned from sensor API calls and handle failures appropriately.

// Note: This is simplified code without proper error handling

// please refer to samples included in the Windows 7 SDK or the

// Windows 7 Sensor Development Kit for complete example code

//

// Description:

// Example code that shows how to create an ISensorManager

// object, and to get the first ALS on the system

// Include file for the Sensors API

#include <sensors.h>

// Declare the ISensorManager object

CComPtr<ISensorManager> pSensorManager;

// Create the ISensorManager object

pSensorManager.CoCreateInstance(CLSID\_SensorManager);

// Declare the ISensorCollection object returned

// by ISensorManager::GetSensorsByType()

CComPtr<ISensorCollection> pALSCollection;

// Get all the ALS sensors on the system

pSensorManager ->GetSensorsByType(

SENSOR\_TYPE\_AMBIENT\_LIGHT,

&pALSCollection);

// Declare the ISensor object that we will

// use for the first ALS on the system

CComPtr<ISensor> pALSSensor;

// Obtain the first ALS on the system from the collection

// returned by ISensorManager::GetSensorsByType()

pALSCollection->GetAt(0, &pALSSensor);

### Requesting Sensor Permissions

To obtain data from a sensor, the calling program must have permissions to access that sensor. If the program does not have permissions to that sensor, the program can request that the user grant permissions.

The following code shows how to use the ISensorManager::RequestPermissions() function:

// Note: This is simplified code without proper error handling

// please refer to samples included in the Windows 7 SDK or the

// Windows 7 Sensor Development Kit for complete example code

//

// Description:

// Declare the ISensorManager object

CComPtr<ISensorManager> pSensorManager;

// Create the ISensorManager object

pSensorManager.CoCreateInstance(CLSID\_SensorManager);

CComPtr<ISensorCollection> pALSCollection;

// Get all the ALS sensors on the system

pSensorManager ->GetSensorsByType(

SENSOR\_TYPE\_AMBIENT\_LIGHT,

&pALSCollection);

// Request permissions for all ALS on the system

HRESULT hr;

hr = pSensorManager->RequestPermissions(

0, // Owner window

pALSCollection, // Collection of sensors requiring permissions

TRUE); // Modal flag

// Now, the application should handle the return value from the

// permissions call, and/or handle individual ISensorEvents::OnStateChanged()

// events

### Sensor Event Handling

Programs may want to dynamically handle sensor enumeration. Your program handles events that are raised when sensors enumerate on the system through ISensorManagerEvents by implementing this callback interface. In this way, sensors can be connected to when they enumerate on the system after your program has started.

The following code shows how to handle the ISensorManagerEvents::OnSensorEnter() events:

// Note: This is simplified code without proper error handling

// please refer to samples included in the Windows 7 SDK or the

// Windows 7 Sensor Development Kit for complete example code

//

// Description:

// Declare the ISensorManager object

CComPtr<ISensorManager> pSensorManager;

// Create the ISensorManager object

pSensorManager.CoCreateInstance(CLSID\_SensorManager);

…

// Declare the event sink object which

// implements ISensorManagerEvents. This object needs to

// be valid while the event sink is set, in this case let’s

// assume this is a class member

CSensorManagerEventSink mySensorManagerEventSink;

…

// Set the event sink

pSensorManager->SetEventSink(&mySensorManagerEventSink);

After the event sink is set, the sensor event sink object is called (specifically, the override of ISensorManagerEvents::OnSensorEnter() ) when sensors appear on the system.

## Using ISensor and ISensorEvents

Now that we know the basics of querying for sensors and requesting permissions for sensors, let’s consider how we use the ISensor and ISensorEvents interfaces to interact with individual sensors.

### Sensor Properties

The ISensor interface is used to examine sensor properties, support data fields, and hook up to events. Properties describe the sensor and its capabilities. Example properties include the following:

* SENSOR\_PROPERTY\_PERSISTENT\_UNIQUE\_ID uniquely identifies the sensor as enumerated on the system.
* SENSOR\_PROPERTY\_MANUFACTURER indicates the manufacturer of the sensor hardware.
* SENSOR\_PROPERTY\_MODEL indicates the model that corresponds to the sensor hardware.
* SENSOR\_PROPERTY\_SERIAL\_NUMBER specifies the serial number for the sensor device.
* SENSOR\_PROPERTY\_CONNECTION\_TYPE specifies how the sensor hardware is connected to the computer.
* SENSOR\_PROPERTY\_ACCURACY specifies the accuracy for each of the data fields exposed by the sensor.
* SENSOR\_PROPERTY\_RESOLUTION specifies the resolution for each of the data fields exposed by the sensor.

These properties are returned as PROPVARIANT objects. The SENSOR\_PROPERTY\_CONNECTION\_TYPE property is of particular interest for ambient light sensors because programs may be interested only in the sensors that are built into the enclosure of the computer. This is because ambient light sensors connected then report the actual lighting conditions of the computer, not lighting conditions corresponding to a separate device (such as a device connected by USB to the computer).

The following example code shows how to get all ambient light sensors on the system and then build up a collection of light sensors that are built into the computer:

// Note: This is simplified code without proper error handling

// please refer to samples included in the Windows 7 SDK or the

// Windows 7 Sensor Development Kit for complete example code

//

// Description:

// Declare the ISensorManager object

CComPtr<ISensorManager> pSensorManager;

// Create the ISensorManager object

pSensorManager.CoCreateInstance(CLSID\_SensorManager);

// Declare the ISensorCollection object returned

// by ISensorManager::GetSensorsByType()

CComPtr<ISensorCollection> pALSCollection;

// Get all the ALS sensors on the system

pSensorManager->GetSensorsByType(

SENSOR\_TYPE\_AMBIENT\_LIGHT,

&pALSCollection);

// Declare the ISensorCollection object

// to store integrated ALS

CComPtr<ISensorCollection> pIntegratedALS;

// Create the ISensorCollection object

pIntegratedALS.CoCreateInstance(CLSID\_SensorCollection);

// Get the count for all ALS on the system

ULONG count;

pALSCollection->GetCount(&count);

// Loop through the ALS collection and add the ALS

// which are integrated into the enclosure of the computer

// into the integrated ALS collection

for(int index = 0; index < count; index++)

{

// Declare and initialize the PROPVARIANT

PROPVARIANT connectionType;

PropVariantInit(&connectionType);

// Get the current sensor for the loop iteration

// we're on

CComPtr<ISensor> computerCurrentSensor;

pALSCollection->GetAt(index, &computerCurrentSensor);

// Get the connection type for the ALS sensor

computerCurrentSensor->GetProperty(

SENSOR\_PROPERTY\_CONNECTION\_TYPE,

&connectionType);

// Check for the computer integrated connection type

if(connectionType.uiVal == SENSOR\_CONNECTION\_TYPE\_computer\_INTEGRATED)

{

pIntegratedALS->Add(computerCurrentSensor);

}

PropVariantClear(&connectionType);

}

// Now use sensors from the pIntegratedALS collection

### Subscribing to Events

Now that we have shown how to filter a list of ambient light sensors, we can set up an event sink to handle notifications. In this case, we’re going to use a common event sink object to handle data coming from all light sensors on the system.

The following example code shows how to set the event sink for ambient light sensors. It builds on the last section of code:

// Note: This is simplified code without proper error handling

// please refer to samples included in the Windows 7 SDK or the

// Windows 7 Sensor Development Kit for complete example code

//

// Description:

// Declare the ISensorCollection object to store

// integrated ALS. Here we’ll assume that this

// collection is already populated

CComPtr<ISensorCollection> pIntegratedALS;

...

// An object which implements ISensorEvents. Here,

// we’ll assume that this event sink object is a

// member of the class which this example could

// belong to. Because it’s a member, it will remain a

// valid object

CALSEventSink m\_ALSEventSink;

...

// Get the count for all ALS on the system

ULONG count;

pALSCollection->GetCount(&count);

// Loop through the ALS collection and set the event sink

// for each sensor

for(int index = 0; index < count; index++)

{

// Get the current sensor for the loop iteration

// we're on

CComPtr<ISensor> computerCurrentSensor;

pIntegratedALS->GetAt(index, &computerCurrentSensor);

computerCurrentSensor->SetEventSink(&m\_ALSEventSink);

}

### Handling Events and Interpreting Ambient Light Sensor Data

Based on the example code we just looked at, our event sink object is called when sensor events for ambient light sensors are raised. Generally, your object that implements ISensorEvents should implement the following functionality for event handling:

* ISensorEvents::OnStateChanged()handles the sensor’s state changing from ready to standby, and so on. For more information, see the SensorState enumeration in the SDK documentation.
* ISensorEvents::OnDataUpdated()hHandles new data coming from the sensor.
* ISensorEvents::OnEvent()handles non-data events including custom events defined by hardware vendors.
* ISensorEvents::SensorLeave()handles the scenario where the sensor is no longer present on the system.

Note The ISensor interface pointer passed to ISensorEvents::SensorLeave()for the sensor that left the system is no longer functional and should be set to NULL at this point.

When new data is available, the ISensorEvents::OnDataUpdated() method is called, and your program can use the light sensor data.

The following example code shows how to retrieve ambient light sensor data in your event sink implementation, and how to interpret the basic meaning of the data as ambient lighting conditions:

// Note: This is simplified code without proper error handling

// please refer to samples included in the Windows 7 SDK or the

// Windows 7 Sensor Development Kit for complete example code

//

// Description:

// Handler for new ALS data

// Override of ISensorEvents::OnDataUpdated

// a part of an event sink implementation for ISensorEvents

STDMETHODIMP CALSEventSink::OnDataUpdated(

ISensor\* pSensor,

ISensorDataReport\* pNewData)

{

// Declare and initialize the PROPVARIANT

PROPVARIANT lightLevel;

    PropVariantInit(&lightLevel);

// Get the sensor reading from the ISensorDataReport object

    pNewData->GetSensorValue(

        SENSOR\_DATA\_TYPE\_LIGHT\_LEVEL\_LUX,

        &lightLevel);

// Extract the float value from the PROPVARIANT object

float luxValue = lightLevel.fltVal;

// Normalize the light sensor data

double lightNormalized = ::log10(luxValue) / 5.0;

// Handle UI changes based on the normalized LUX data

// which ranges from 0.0 - 1.0 for a lux range of

// 0 lux to 100,000 lux, this method represents such a

// handler that would be implemented in your application

    UpdateUI(lightNormalized);

PropVariantClear(&lightLevel);

return S\_OK;

}

# Light Sensor Data Types

Three common light sensor data types may be exposed by typical ambient light sensors. These sensor data types for ambient light sensors are based on the definitions in the ACPI 3.0b specification for ambient light sensors (section 9.2).

The following is a brief overview of the common light sensor data types, and how they are defined in the Windows 7 sensor platform.

## **Illuminance**

Illuminance is a measure of the overall magnitude of the ambient light level. The units for illuminance are lux. The sensor platform definition for illuminance is:

SENSOR\_DATA\_TYPE\_LIGHT\_LEVEL\_LUX

## **Color Temperature**

Color temperature is a measure of the warmth or coolness of light. Color temperature corresponds to the color of light given off by a black body radiator heated to a particular temperature. The corresponding units for color temperature are degrees Kelvin. The sensor platform definition for color temperature is:

SENSOR\_DATA\_TYPE\_LIGHT\_TEMPERATURE\_KELVIN

## **Chromacity (also known as Chromaticity)**

Chromacity is a spectral characteristic of light based on a 2‑D coordinate system (CIE Yxy color model). For more information about chromacity, see the ACPI specification. The sensor platform definition for chromacity is:

SENSOR\_DATA\_TYPE\_LIGHT\_CHROMACITY

# Handling Data from Multiple Light Sensors

To produce the most accurate approximation of the current lighting condition, data from multiple ambient light sensors should be used. Because ambient light sensors can be partly or fully obscured by shadows or other objects that cover the sensor, multiple sensors placed some distance apart can provide a much better approximation of the current lighting conditions.

To keep track of the data coming from multiple sensors, the following two techniques can be used:

* The most recent data values for each ambient light sensor should be retained, along with the time stamp from the sensor data report for each of these readings. The last ISensorDataReport for each sensor reading could be retained and could provide both values for later reference. By referring to the time stamp for each sensor data report, the data can be “aged.” For example, if the data is more than 2 seconds old, it could be omitted. Based on the newer sensor data values, the highest reading could be used because the corresponding sensor would be presumed not to be obscured.
* The simple but inaccurate approach is to use the last ambient light sensor value reported. This implementation would not be optimal because the values from multiple sensors are not compared to one another to obtain the most accurate result. This method is not recommended.

# Conclusion

Sensors are becoming more prolific in electronics and computers. As sensor hardware becomes more mainstream, the opportunities for context-aware computing become more attainable.

In Windows 7, these opportunities can be realized because of the new Windows Sensor and Location platform, which provides native support for these devices. Sensor programs for Windows 7 only need to know what types of sensor hardware are of interest. They do not require specific code for proprietary sensor-related APIs or driver architectures. Because Windows 7 provides driver support for ACPI-implemented ambient light sensors, the opportunities for light-aware programs are abundant. Finally, having programs use ambient light sensors can have a huge effect on overall usability in a wide variety of lighting conditions.

# Call to Action

If you are interested in making your program light-aware or are interested in investigating other sensor scenarios, you can do several things to get started:

* Install Windows 7.
* Download and install the Windows 7 SDK.
* Get a Windows 7 Sensor Development Kit.

# For More Information

For more information about sensor devices, see the official site for sensor devices at:

<http://www.microsoft.com/whdc/sensors>

For more information about ACPI light definitions, see the 3.0b specification at:

<http://acpi.info>

# Appendix A

To view the SDK documentation

1. Click Start, click All Programs, click Microsoft Windows SDK v7.0, and then click Windows SDK Documentation.

2. Browse to the Sensor API documentation ($\Win32 and COM Development\Sensor API).

To build the sensor samples

1. Open a Windows 7 SDK command window.

2. Change to: Samples\winui\Sensors.

3. Browse to subdirectories, and open the individual projects in Visual Studio 2008 (full or express products).

4. Build and run the samples.

To set up your project in Visual Studio

1. Make sure that you have Visual Studio 2008 installed (full or express products).

2. Open up Visual Studio and create your Win32 project (MFC, ALT, Win32, and so on).

3. Go to your current project, and then right click and select Properties.

In the Configurations list, click All Configurations.

Click the C++ node, and then click General settings.

In the Additional Include Directories box, enter Add $(ProgramFiles)\Microsoft SDKs\Windows\v7.0\Include.

Click the Linker node, and then click Input.

In the Additional Dependencies box, enter $(ProgramFiles)\Microsoft SDKs\Windows\v7.0\Lib\SensorsApi.lib.

4. Make sure that the following is included in all files that use the sensor API individually or in a common include file, such as stdafx.h, for a project with precompiled headers:

#include <sensors.h>

5. Make sure that you are calling CoInitialize() or CoInitializeEx() somewhere in your program to initialize the COM runtime

Now that your project is created and configured to use the sensor API, build and run the project to make sure headers are resolved. Now you’re ready to start writing sensor code!

# Appendix B

## Sensor API Concepts

To understand how to use the sensor platform it is helpful to first learn about the high-level concepts that are relevant to this platform.

### Sensor Categories

Sensor categories are used to group sensors that are likely to report the same type of data, or are otherwise related in some way. An example of a sensor category would be location sensors, which usually report latitude and longitude.

### Sensor Types

Sensor types correspond to a specific type of sensor hardware or a sensor that measures a phenomenon in a particular way. Example types of sensors would include GPS sensors, which use satellite triangulation to determine location, and IP resolver sensors, which use the IP address of the computer to resolve the current location. These sensors both belong to the location category, because they report location information.

### Sensor Data Types

Sensor data types correspond to measurements that sensors report. Sensor data types are a union of:

* The phenomenon being measured (Example: altitude).
* The dataum point relative to which the measurements correspond (Example: sea level).
* The units in which the measurements are reported (Example: meters).

Given these criteria, the sensor and location platform defines the following for altitude:

SENSOR\_DATA\_TYPE\_ALTITUDE\_SEALEVEL\_METERS

With this brief introduction to sensor platform concepts, we can now consider how to get started programming with the sensor API.