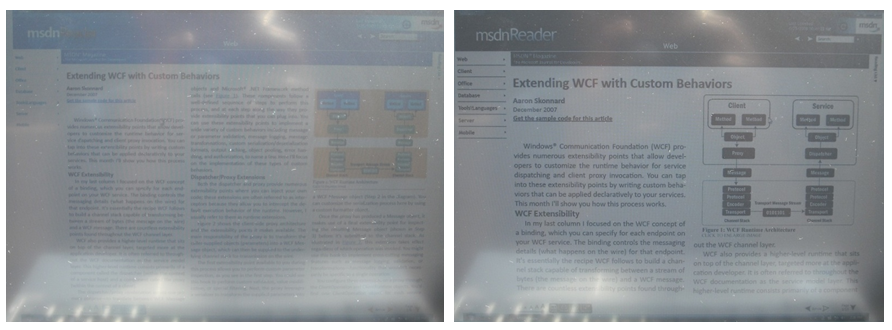
**Windows 7 Sensor and Location .NET Interop Sample Library**

The Windows 7 Sensor and Location platform enables your computer and applications to adapt to their current environment and change the way they look, feel, or behave. For example, an application that is using a light sensor can change the brightness and contrast of its displayed content. With location sensors, applications can know their current location, enabling many different scenarios such as content localization and functionality.

The Sensor and Location platform provides a standard way to integrate sensor and location devices with Windows 7, and provides a standard programming model for applications to take advantage of these devices. [Hardware manufacturers can learn](http://blogs.msdn.com/gavingear/archive/2009/02/05/want-to-write-a-sensor-driver-windows-7-beta-wdk-docs-are-live-on-msdn-com.aspx) how to write sensor and location drivers by installing the [Windows 7 WDK](http://www.microsoft.com/whdc/devtools/WDK/default.mspx). Developers can learn how to write location-aware and sensor-enabled applications by installing the [Windows 7 Beta SDK](http://www.microsoft.com/downloads/details.aspx?FamilyID=A91DC12A-FC94-4027-B67E-46BAB7C5226C&displaylang=en).

Some may ask, **why use the Windows 7 Sensor and Location Platform?** To answer that question, we need to understand that most of the applications that we use on a daily bases were designed for “normal” environmental conditions, for example, under indoor standard lighting conditions. However, nowadays laptops offer higher mobility than ever before, allowing us to use our laptops in different locations with different environments, switching from bright outdoor light into darker indoor light. Wouldn’t it be nice to have an application that adjusts its look, feel, and functionality to the change in the environment? Ambient light sensors, for example, can allow your computer to automatically adjust your screen's brightness based on the current lighting conditions. Furthermore, developers can also enable applications to optimize their content for readability, making your application easier to use in a range of different operating environments.

The example below shows an updated version of the MSDN Reader. This version changes the way the application looks depending upon the amount of light the Ambient Light Sensor detects. On the left is how the application looks under “normal” indoor lighting conditions. On the right, the application reduces the use of colors to increase contrast and increases the font size, making the displayed content (the actual text displayed) of the application more readable under the brighter lighting conditions.



The Windows 7 Beta SDK provides tools, documentation, and examples for writing drivers for sensor devices as well as using the Sensor and Location platform in building Windows 7 applications. As is true for most Windows 7 APIs, the Sensor and Location API is a native one. There are some examples that display managed code applications using a very thin interop layer. However, this ***Windows 7 Sensor and Location .NET Interop Sample Library*** offers a broader and more “.NET like” programming model with strongly typed objects and an easy extensibility model for sensor and data reports.

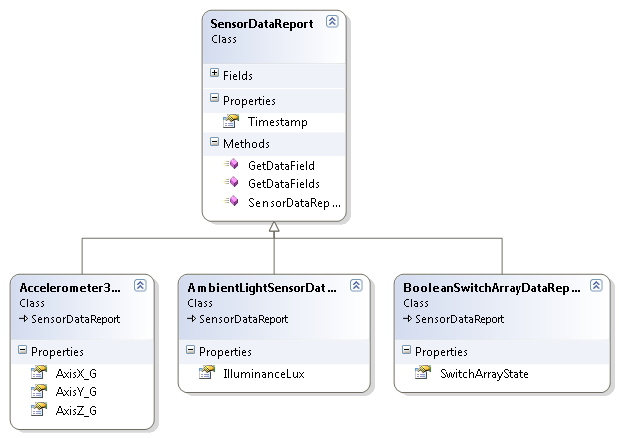
Let’s review the ***Windows 7 Sensor and Location .NET Interop Sample Library*** architecture and mention the most important classes.

|  |  |
| --- | --- |
| To start working with sensors, you must first obtain one. You can do this using the ***SensorManager*** class whose main purpose in life is to expose sensor classes (for example light or accelerometer sensor classes) to the developer. Using the ***SensorManager,*** developers can: | |
| * **Get an array of sensors by CategoryID**. ***CategoryID*** is a GUID that defines a well known specific class of sensors such as Light Sensors, Accelerometer Sensors, Motion Sensors, or Location. You can find the complete list of well known sensors in the ***SensorCategories*** class. * **Get an array of sensors by TypeID**. ***TypeID*** is a specific type of a sensor that is related to a specific class of sensors. For example, Type of ***LocationGPS*** is part of the Location Sensors class, and ***Accelerometer3d*** and ***Accelerometer2d*** are related to the Accelerometer Sensors class. You can find the complete list of sensor Types in the ***SensorCategories*** class. * **Get a specific Sensor** using the sensor's GUID. |  |

* **GetAllSensors,** which retrieves all the sensors that are in the register. You can use any sensors from that list as long as you have permission to do so. The end user will have to enable each sensor for the application to use.
* **Show the** **permission dialog box** to provide users with a way to enable sensors.
* **Notify** the application of a sensor's availability with the ***SensorAttached*** event.

Once you have obtained a sensor from the ***SensorManager,*** the application can retrieve data using a ***SensorDataReport*** class, which is an abstract base class for all ***DataReports***. The native API includes only one generic ***DataReport***, defined by a **C** **Union** **type,** which, come to think of it, makes perfect sense to use for native applications since generated data reports arriving from different sensor categories will most likely have different data fields in the reports. For example, a light sensor produces different data than a location sensor does. However, a union has no formal (or informal) representation in .NET since a native union is based upon a fixed memory allocation and the “physical” memory size of parameters, which have no equivalent in the .NET Framework. Therefore, our goal is to create a strong typed .NET class for each sensor and for each sensor data report, enabling easy binding to the .NET property system. The ***SensorDataReport*** is the base class for sensor type-tailored data reports (for example, **Accelerometer3DReport**). When developers implement support for a custom sensor, they usually create a set of two classes: a **Sensor**-derived class, and a **SensorDataReport**-derived class. The ***SensorDataReport*** provides base methods, such as querying a property by **PropertyKey,** which are intended to be overridden by type-safe properties in derived classes (for example, **AxisY\_G** – float in **Accelerometer3DReport**).

The follow diagram illustrates three derived data report classes that we implemented in the ***Windows 7 Sensor and Location .NET Interop Sample Library***. An Accelerometer3DDataReport, an AmbientLightSensorDataReport, and a BooleanSwitchArrayDataReport.



We also implemented a ***LatLongLocationReport*** and a ***CivicAddressLocationReport*** that you can find under the Location Providers in the Windows 7 Sensor and Location .NET Interop Sample Library.

Let’s examine few lines of code that illustrate the process of reading the amount of light by using the ***AmbientLightSensor*** sensor and data report. To do so, a developer will need to:

1. Get an ***AmbientLightSensor*** class from the ***SensorManager***
2. Get a data report from the ***AmbientLightSensor*** class. Note that *GetDataReport* returns a generic ***SensorDataReport***

The following code snippet shows how you can obtain a light sensor from the ***SensorManager*** and use that light sensor to get an ***AmbientLightSensorDataReport***.

//obtain a light sensor

AmbientLightSensor[] lightSensors = SensorManager.GetSensorsByType<AmbientLightSensor>();

if (lightSensors.Length > 0)

{

\_lightSensor = lightSensors[0];

\_lightSensor.ReportInterval = \_lightSensor.MinReportInterval;

}

…

//use the light sensor to get en AmbientLightSensorDataReport

AmbientLightSensorDataReport lightReport = (AmbientLightSensorDataReport)\_lightSensor.GetDataReport();

\_illuminanceLux = lightReport.IlluminanceLux;

|  |  |
| --- | --- |
| Developers can do a lot more with a given sensor then just retrieve a data report. The Sensor base abstract class contains a comprehensive set of properties that provide general info like *FriendlyName*, *SensorDescription*, *SensorID*, *SensorSerialNumber*, and *SensorState*. There are also specific properties such as ReportInterval to set the intervals between each time a sensor device generates a report, and MinReportInterval to set the minimum time between reports. In addition, the Sensor class exposes a set of events including DataUpdated that is triggered when a new data report is ready, and OnStateChanged that is triggered when the sensor state has changed. The Sensor class exposes additional properties that you are welcome to explore. The following diagram shows the Sensor class and three derived classes we created for the purpose of the Sensors and Location Managed Library Sample. |  |

The Sensor and Location Managed Library Sample includes two demos: the MSDN Reader and Marble Sample. The latter is an XNA application that requires you to download the [XNA Game Studio 3.0](http://www.microsoft.com/downloads/details.aspx?familyid=7D70D6ED-1EDD-4852-9883-9A33C0AD8FEE&displaylang=en). This demo shows how to use the ***Accelerometer3D*** sensors. The MSDN Reader uses the ***AmbientLightSensor*** to change the size and color of the fonts in the reader.

We will also upload an updated version of the XNA Car Racing Starter Kit that uses the ***Accelerometer3D*** sensors to control the direction and speed the car is going and the ***AmbientLightSensor*** to control the color of the sky.



In a future post, we will further explore the architecture we chose for the Windows 7 Sensor and Location .NET Interop Sample Library and expand your understanding of how to create your own set of sensors.