SensorManager

Guide

**Legal Disclaimers and Copyrights**

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, Intel® Core™ vPro™ processor family are trademarks of Intel Corporation in the United States and/or other countries.

\*Other names and brands may be claimed as the property of others

Contents

[1 Code 9](#_Toc22134969)

[1.1 Add Sensor Data for other plugins 9](#_Toc22134970)

[1.1.1 Create a local SensorManager 9](#_Toc22134971)

[1.1.2 Understanding Sensor 9](#_Toc22134972)

[1.1.3 Create a Sensor 10](#_Toc22134973)

[1.1.4 Link Sensor to SensorManger 11](#_Toc22134974)

[1.2 Get Sensor Data from other plugins 12](#_Toc22134975)

[1.2.1 Subscribe to Sensor Updates 12](#_Toc22134976)

# Walkthrough

## Add Sensor Data for other plugins

Intel Unite® solution provides a way for plugins to access sensor information. Mock data will be used for this illustration on how to accomplish Plugin-to-Sensor communication.

Requirements:

* Minimum Plugin Project Setup
* Sensor Data (In this example the data is mocked)
* Understanding the Sensor Class

The Sensor class is used to store information about the sensor as well as the data it collects.

#### FriendlyName

A name which is easy to understand.

#### UniqueName

A Unite Unique Name, for example "UnitePlugin\_Temp\_Probe\_1".

#### Id

A unique GUID.

#### ModuleId

The GUID of the plugin the sensor was generated from.

#### Type

Type of sensor [0-127 are reserved types, 128-255 are custom]. The reserved types are defined in Intel.Unite.Common.Sensor.UniteSensorType.

#### KeyValueProperties

This is where the data of the sensor is held. A sensor can have multiple readings. A weather station sensor might have air pressure, temperature, and humidity readings.

#### Expiration

The moment when the data is no longer valid. Data will be kept for a maximum of 24 hours. Intel Unite® solution assumes that if the sensor is active, then data will be regularly updated. If the sensor doesn’t update, the data will expire and be removed.

### Create a Plugin SensorManager

Used to create a method to attach to a plugin specific event which will route to SensorManager.Set.

Code 1 - PluginSensorManager

|  |
| --- |
| public class PluginSensorManager  {  public ISensorManager SensorManager { get; }  public PluginSensorManager(ISensorManager sensorManager)  {  SensorManager = sensorManager;  }  public void UpdateSensorData(object sender, SensorArgs args)  {  SensorManager.Set(args.Sensor);  }  } |

### Create a Sensor

A sensor must regularly update, otherwise its data will expire and other plugins will not have access to that data. The sensor created must generate the Sensor class for consumption by the SensorManager.

Code 2 - MockSensor

|  |
| --- |
| public class MockSensor  {  private const int \_maxChange = 3;  private static readonly Random \_random = new Random();  private static readonly Timer \_aTimer = new Timer(2000);    private static int Temp { get; set; } = 72;  public static event EventHandler<SensorArgs> UpdateSensorData;  public const string UniqueName = "UnitePlugin\_Temp\_Probe\_1";    public static void Start()  {  \_aTimer.Elapsed += OnTimedEvent;  \_aTimer.AutoReset = true;  \_aTimer.Start();  }  public static void Stop()  {  \_aTimer.Elapsed -= OnTimedEvent;  \_aTimer.AutoReset = false;  \_aTimer.Stop();  }  private static void OnTimedEvent(object source, ElapsedEventArgs e)  {  Temp += \_random.Next(\_maxChange) - \_random.Next(\_maxChange);  UpdateSensorData?.Invoke(source, new SensorArgs(GetTempSensor(Temp)));  }  private static Sensor GetTempSensor(int temp)  {  return new Sensor  {  FriendlyName = "Unite Plugin Temperature",  Expiration = DateTime.Now.AddHours(24),  Id = Guid.NewGuid(),  KeyValueProperties = new List<SensorKeyValue> {new SensorKeyValue{Key = "Value", Value = temp.ToString(), ValueType = SensorValueType.Int}},  ModuleId = Guid.NewGuid(),  Type = (int)UniteSensorType.Temperature,  UniqueName = UniqueName  };  }  } |

### Configure the Sensor

Setup the links between the Sensor, Plugin SensorManager, and the Common SensorManager

Code 3 - SensorConfig

|  |
| --- |
| public static class SensorConfig  {  public static void Setup()  {  SetupPluginSensorManager();  SetupLocalSensorUpdates();  }  private static void SetupLocalSensorUpdates()  {  MockSensor.UpdateSensorData += UnitePluginConfig.PluginSensorManager.UpdateSensorData;  }  private static void SetupPluginSensorManager()  {  UnitePluginConfig.PluginSensorManager = new PluginSensorManager(UnitePluginConfig.RuntimeContext.SensorManager);  }  } |

### Add sensor methods to Load

The load method is called when a plugin is loaded by the Unite Core and is the best place to call configuration methods.

Code - 4 Load Sensor

|  |
| --- |
| public override void Load()  {  UnitePluginConfig.RuntimeContext = RuntimeContext;  SensorConfig.Setup();  MockSensor.Start();  } |

## Get Sensor Data from other plugins

Intel Unite® solution provides a way for plugins to communicate sensor information from one plugin to another through the SensorManager Interface.

Requirements:

* Minimum Plugin Project Setup
* A Sensor is added by a plugin

### Process Sensor Data

How the plugin will handle sensor data updates. In this example a toast message is created every time the temperature is updated.

Note: Any method passed to the unite core to run must be public, non-static, and be in a class the inherits MarshalByRefObjectBase.

Code 5 SensorHandler

|  |
| --- |
| public class MockSensorHandler : MarshalByRefObjectBase  {  public const int VisibilityTime = 10;  public void ProcessData(object sender, SensorArgs e)  {  if (e.Sensor.UniqueName == MockSensor.UniqueName)  {  UnitePluginConfig.RuntimeContext.DisplayManager.TryShowToastMessage(e.Sensor.KeyValueProperties  .FirstOrDefault(x => x.Key == "Value")?.Value, VisibilityTime);  }  }  } |

### Subscribe to Sensor Updates

Link the sensor handler to the sensor added event. The sensor added event will trigger every time SensorManager.Set is called.

Code 6 Setup Sensor Updates

|  |
| --- |
| public static class SensorConfig  {  public static void Setup()  {  SetupCoreUpdates();  }  private static void SetupCoreUpdates()  {  UnitePluginConfig.RuntimeContext.SensorManager.SensorAdded += new MockSensorHandler().ProcessData;  }  } |