



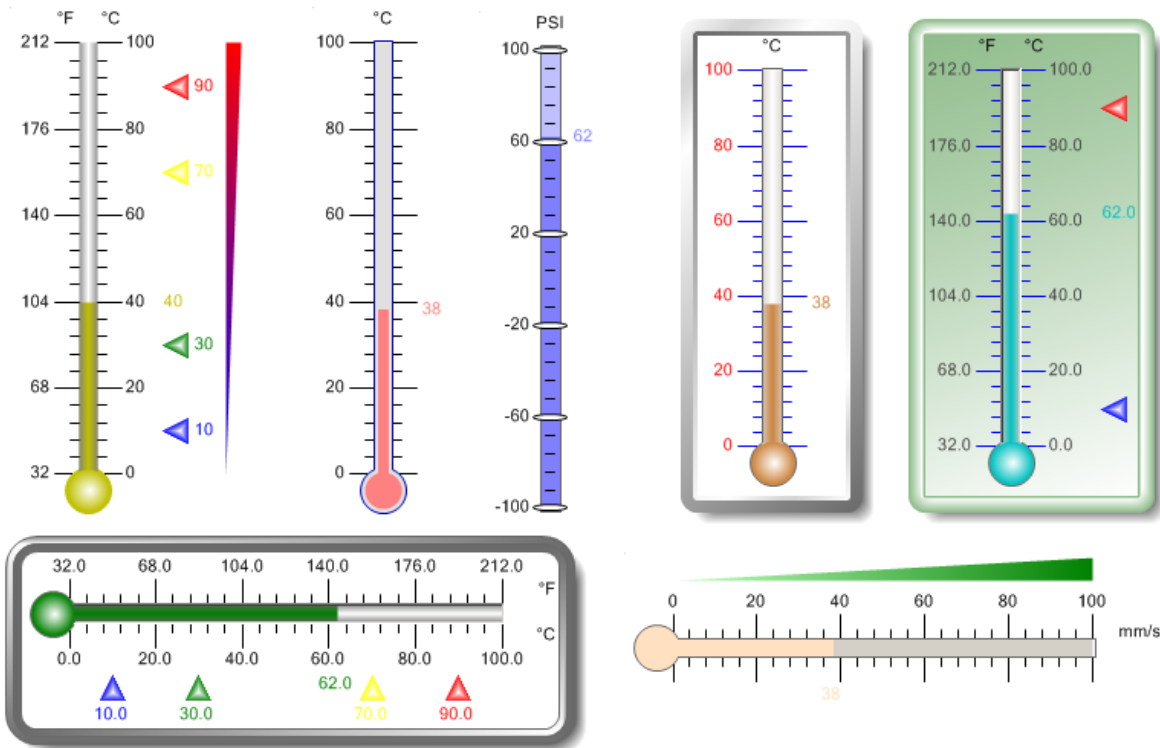
## Using Temperature Meter .Net Control

### 1. Overview

The Temperature Meter .Net Windows Form control allows users to show the temperature in a visual meter. This control can be used in heats control and temperature monitor system or other monitoring systems. This product is very powerful for designers to configure different kinds of temperature meters. There are a rich group of the properties which enable people to design any temperature meter to meet their requirements. All these properties will be addressed in the following sections.

### 2. DAS\_Net\_Temperature

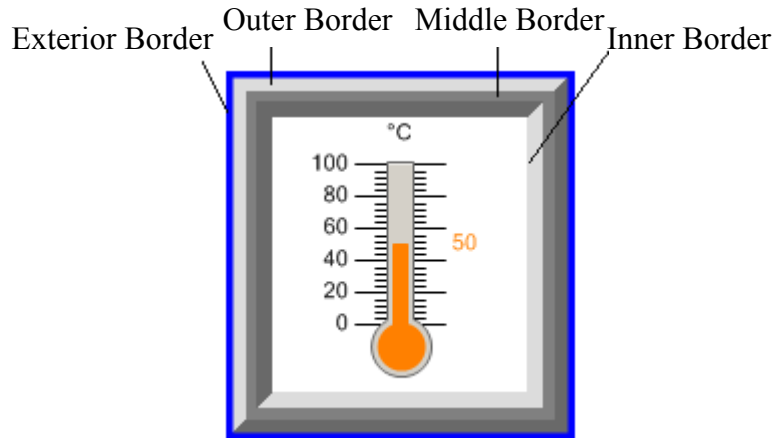
The designers can use the interface properties and methods to implement many kinds of temperature meters, the following picture shows some styles which can be configured.



### Properties:

### Border Properties:

To make the control more intuitive and vivid look-and-feel, a group of border properties are provided for the designers to configure the control's border style. There are four border layers, i.e., Exterior Border layer, Outer Border layer, Middle Border layer and Inner Border layer,



We can configure different color and layer length for each layer. Especially for Outer-Border and Inner-Border, the light color and the dark color can be configured to generate different gradient 3D-Border feeling.

Property *DAS\_BorderStyle BorderShape*

Specify which border shape (Rectangle or Round Rectangle) is used to draw the border of control. There are three definitions, i.e., *BS\_Rect* defines Rectangle Shape, *BS\_RoundRect* defines Round Rectangle Shape (The size of the round corners is defined by the property *RoundRadius*).

Property *int BorderExteriorLength*

Specify the border length of the Exterior Border, this layer is drawn using the color defined by the property *BorderExteriorColor*. The default length of this layer is zero.

Property *Color BorderExteriorColor*

Specify the color to render the exterior border layer.

Property *int OuterBorderLength*

Specify the border length of the outer border, this layer is drawn using the colors defined by the property *OuterBorderDarkColor* and *OuterBorderLightColor*. This layer can be rendered in different gradient modes which is defined by the property *BorderGradientType*.

Property *Color OuterBorderDarkColor*

Specify the dark color of the outer border that is hidden from the light.

Property *Color OuterBorderLightColor*

Specify the light color of the outer border that faces the light.

Property *int InnerBorderLength*



Specify the border length of the inner border, this layer is drawn using the colors defined by property *InnerBorderDarkColor* and *InnerBorderLightColor*. This layer can be rendered in different gradient modes which is defined by the property *BorderGradientType*.

Property Color InnerBorderDarkColor

Specify the dark color of the inner border that is hidden from the light.

Property Color InnerBorderLightColor

Specify the light color of the inner border that faces the light.

Property int MiddleBorderLength

Specify the length of the middle border layer, this layer is rendered by property *MiddleBorderColor*.

Property Color MiddleBorderColor

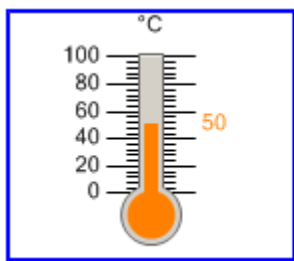
Specify the color to render the middle border layer.

Property int RoundRadius

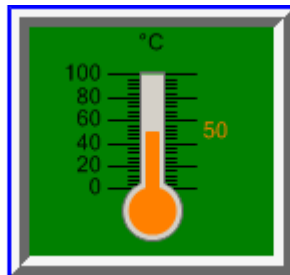
Specify the round corner size of the round-rectangle shape.

### Border Gradient Properties:

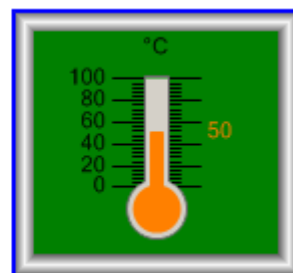
There are six approaches defined in *DAS\_BorderGradientStyle* to render the control border,



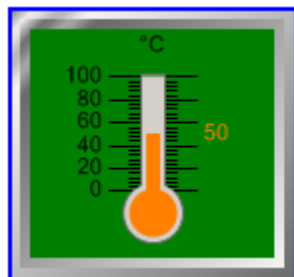
BGS\_None



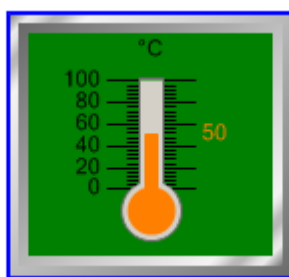
BGS\_Flat



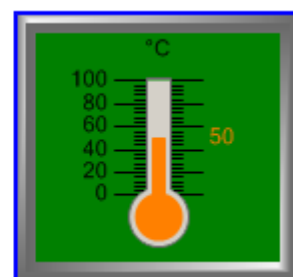
BGS\_Ring



BGS\_Linear



BGS\_Linear2



BGS\_Path

BGS\_None ---- Ignore the Outer Layer, Middle Layer and Inner Layer, only the Exterior layer is rendered.



- BGS\_Flat ---- All layers are rendered. The left side and top side of the Outer Layer are rendered using *OuterBorderLightColor*, the right side and bottom side of the Outer Layer are rendered using *OuterBorderDarkColor*; The left side and top side of the Inner Layer are rendered using *InnerBorderDarkColor*, the right side and bottom side of the Inner Layer are rendered using *InnerBorderLightColor*;
- BGS\_Ring ---- All layers are rendered. The outer layer is rendered in gradient mode at radial direction from *OuterBorderDarkColor* at the outer most periphery of the outer layer to *OuterBorderLightColor* at the inner most periphery of the outer layer. The inner layer is rendered in gradient mode at radial direction from *InnerBorderLightColor* at the outer most periphery of the inner layer to *InnerBorderDarkColor* at the inner most periphery of the inner layer.
- BGS\_Linear --- All layers are rendered. The outer layer is rendered in gradient mode at the direction defined by property *BorderGradientAngle* from *OuterBorderLightColor* at the one end (defined by property *BorderGradientLightPos1* and *BorderGradientLightPos2*) to *OuterBorderDarkColor* at the other end of the outer layer. The inner layer is rendered in gradient mode at the direction defined by property *BorderGradientAngle* from *InnerBorderDarkColor* at the one end (defined by property *BorderGradientLightPos1* and *BorderGradientLightPos2*) of the inner layer to *InnerBorderLightColor* at the other end of the inner layer.
- BGS\_Linear2 --- Similar to BGS\_Linear. The difference is the definitions of property *BorderGradientLightPos1* and *BorderGradientLightPos2* of inner layer. The inner layer is rendered in gradient mode at the direction defined by property *BorderGradientAngle* from *InnerBorderDarkColor* at the one end of the inner layer to *InnerBorderLightColor* at the other end (defined by property *BorderGradientLightPos1* and *BorderGradientLightPos2*) of the inner layer.
- BGS\_Path ---- All layers are rendered. The outer layer is rendered in gradient mode at the radial direction from *OuterBorderDarkColor* at the outer most periphery to *OuterBorderLightColor* at the center point that is in the outer layer at the direction defined by property *BorderGradientAngle*. The inner layer is rendered in gradient mode at the radial direction from *InnerBorderDarkColor* at the outer most periphery to *InnerBorderLightColor* at the center point that is in the



outer layer at the direction defined by (*BorderGradientAngle* +  $180^\circ$ ).

Property *DAS BorderGradientStyle BorderGradientType*

Specify which border gradient type is applied to render the border of the control.

Property *int BorderGradientAngle*

Specify the gradient angle for *BGS\_Linear*, *BGS\_Linear2* and *BGS\_Path*.

Property *float BorderGradientRate*

Specify the gradient focus size. It should be within 0.0 to 1.0.

Property *float BorderGradientLightPos1*

Specify the gradient light position 1 at the gradient angle. It should be within 0.0 to 1.0. This property just takes effect for *BGS\_Linear* and *BGS\_Linear2*.

Property *float BorderGradientLightPos2*

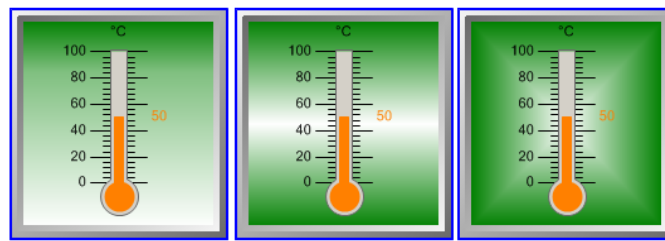
Specify the another gradient light position at the gradient angle. It should be within 0.0 to 1.0. If it is negative, the second gradient light position is ignored. This property just takes effect for *BGS\_Linear* and *BGS\_Linear2*.

Property *float BorderLightIntermediateBrightness*

Specify the color at the middle point between the light position 1 and the light position2. It should be within 0.0 to 1.0. If it's 0, the dark color is applied at middle point, if it's 1.0, the light color is used at the middle point. This property just takes effect for *BGS\_Linear* and *BGS\_Linear2*.

## Background Properties:

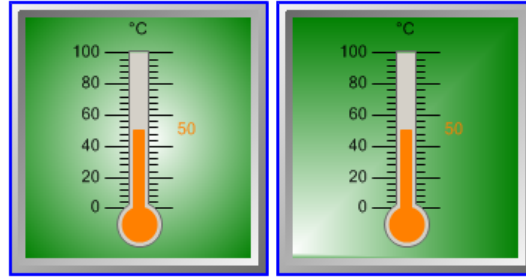
In general, the background of the control can be rendered in gradient mode or normal mode. And the background image (defined by *BackgroundImage*) can be rendered as well using the standard image rendering methods (defined by *BackgroundImageLayout*), like "Tile", "Zoom", "Stretch", "Center" and "None", please refer to the standard background image rendering in MSDN. If *BackgroundImage* is set, the image is first rendered, and then the back color with transparency (defined by *BkTransparency*) is rendered. There are five background gradient types are defined as follows (*DAS\_BkGradientStyle*),



BKGS\_Linear

BKGS\_Linear2

BKGS\_Polygon



BKGS\_Sphere

BKGS\_Shine

The gradient direction for *BKGS\_Linear*, *BKGS\_Linear2* and *BKGS\_Shine* is defined by the property *BkGradientAngle*.

Property Color BackColor

Specify the background color of the control.

Property Image BackGroundImage

Specify the background image of the control. For details, refer to MSDN.

Property ImageLayout BackGroundImageLayout

Specify the background image layout style. For details, refer to MSDN.

Property float BkTransparency

Specify the back color rendering transparency rate. If it's 0.0, there should be no transparency, so if *BackGroundImage* is set, but *BkTransparency* is zero, the image is invisible. If it's 1.0, the image is fully visible, and the gradient back color rendering is not invisible.

Property float BkGradientRate

Specify the gradient focus size, it should be between 0.0 and 1.0.

Property bool BkGradient

Specify whether the background is rendered in gradient mode or normal mode.

Property int BkGradientAngle

Specify the background gradient direction angle for *BKGS\_Linear*, *BKGS\_Linear2* and *BKGS\_Shine*

Property Color BkGradientColor

Specify the background gradient color.

Property DAS BkGradientStyle BkGradientType

Specify the background gradient type.

Property float BkShinePosition

Specify the background gradient shine (focus) position (for *BKGS\_Shine*), it should be between 0.0 and 1.0. In conjunction with *BkGradientAngle*, the gradient shine position can be fully adjusted in the background rendering area.



## Shadow Properties:

Shadow display is a basic feature of the Dragonfly .Net controls.

Property bool ControlShadow

Specify whether to show the shadow or not.

Property int ShadowDepth

Specify the depth of the shadow.

Property Color ShadowColor

Specify the color of the shadow.

Property float ShadowRate

Specify the emission rate of the shadow.

## General Properties:

Property Color ForeColor

Specify the color to draw the units.

Property bool Vertical

Specify whether the meter is drawn vertically or horizontally. Default is True

Property int Precision

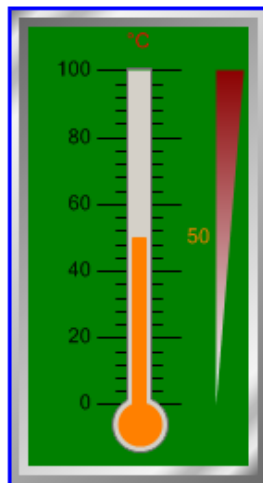
Specify the decimal digit number for the scale texts and actual value display. Default is zero.

Property double Value

Specify the value of the meter.

Property bool ProgressTriangleVisible

Specify the visibility of the progress triangle.





Property Color ProgressLowColor

Specify the color to represent the lower part of the progress triangle.

Property Color ProgressHighColor

Specify the color to represent the higher part of the progress triangle.

Property int ProgressTriangleWidth

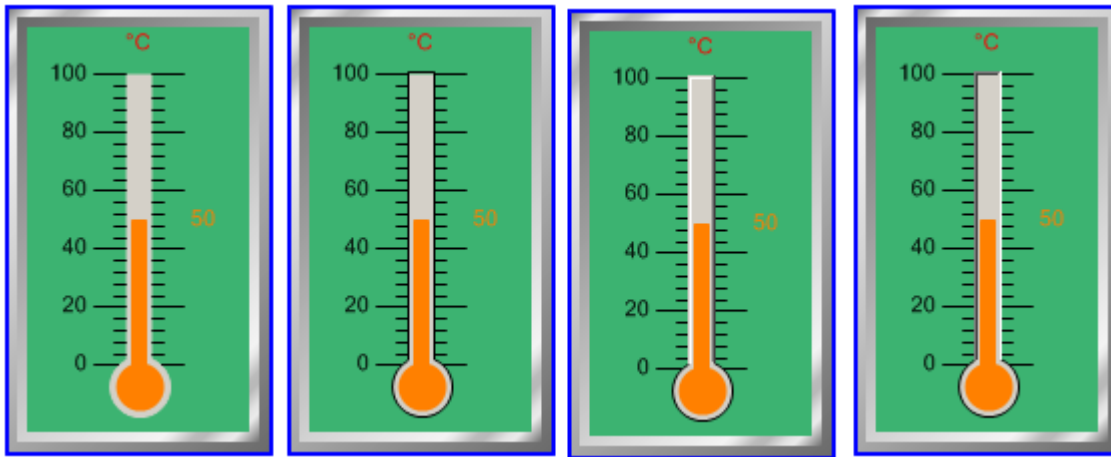
Specify the size of the progress triangle.

Property bool MeterGradient

Specify whether the meter is render in gradient mode or not.

Property DAS MeterBorderStyle MeterBorderType

Specify the border type of the meter. There are four definitions in `DAS_MeterBorderStyle`, i.e.,



MBS\_None

MBS\_Flat

MBS\_Raised

MBS\_Sunken

Property Color MeterColor

Specify the color to render the background of the meter.

Property Color MeterForeColor

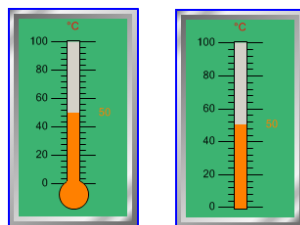
Specify the color to render the foreground of the meter.

Property Color MeterBorderColor

Specify the color to draw the border of the meter.

Property bool BulbVisible

Specify visibility of the meter bulb.





Property bool ActualValueVisible

Specify visibility of the actual value of the meter.

Property int MeterWidth

Specify the width of the meter bar.

Property int MeterInterGap

Specify the gap between the meter bar and inter value bar. Default is 0.

Property int MeterShift

Specify the X shift position of the mater if `Vertical=True`, or Y shift position of the meter if `Vertical = False`.

## Unit Properties:

The meter can support two unit displays, the relationship between the two units are

$$\text{Unit}_2 = \text{SecondUnitRate} * \text{Unit}_1 + \text{SecondUnitOffset}.$$

The following properties are about the units of the meter.

Property string Unit

Specify the first unit of the meter.

Property bool SupportSecondUnit

Specify whether to support the second unit or not.

Property string ScondUnit

Specify the second unit of the meter.

Property double SecondUnitOffset

Specify the second unit offset based the first unit.

Property double SecondUnitRate

Specify the conversion rate of second unit of the meter.

## Scale Properties:

The following properties are used to define the scale.

Property double Max

Specify the maximum value of the scale.

Property double Min

Specify the minimum value of the scale.

Property int TickerNumber

Specify how many tickers are drawn for the scale. The minimum is 2.

Property int SubTickerNumber

Specify how many sub tickers are drawn between two scale tickers.



Property Color TickerLineColor

Specify the color to draw the border of tickers and sub tickers.

Property Color ScaleTextColor

Specify the color to draw scale texts.

Property int TickerWidth

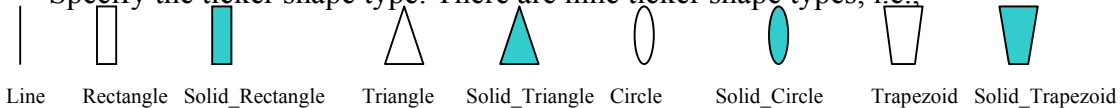
Specify the ticker width if the ticker shape is not “Line”.

Property Color TickerFillColor

Specify the color to fill the ticker if TickerShape is Solid\_Rectangle, Solid\_Triangle, Solid\_Circle or Solid\_Trapezoid.

Property DAS TickerShapeStyle TickerShape

Specify the ticker shape type. There are nine ticker shape types, i.e.,



Property int SubTickerWidth

Specify the sub ticker width if the sub ticker shape is not “Line”.

Property Color SubTickerFillColor

Specify the color to fill the sub ticker if SubTickerShape is Solid\_Rectangle, Solid\_Triangle, Solid\_Circle or Solid\_Trapezoid.

Property DAS TickerShapeStyle SubTickerShape

Specify the sub ticker shape type.

Property int TickerLength

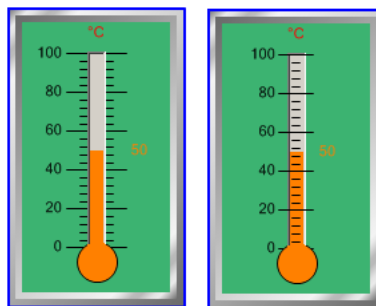
Specify the length of the tickers.

Property int SubTickerLength

Specify the length of the sub tickers.

Property DAS TickerAlignmentStyle TickerAlignment

Specify the scale ticker alignment type. There are two definitions, i.e.,





## Alarm Properties:

The following properties are used to support alarm display and alarm event generation.

Property bool SupportAlarm

Specify whether to support the alarm function or not.

Property double HighAlarmLimit

Specify the high limit that the value of the meter exceeds to be remarked as “Too High” alarm.

Property double LowAlarmLimit

Specify the low limit that the value of the meter exceeds to be remarked as “Too Low” alarm.

Property Color HighAlarmColor

Specify the color to render the foreground of the meter when the value of the meter is higher than *HighAlarmLimit*. This color is also used to render the High Alarm Limit Marker.

Property Color LowAlarmColor

Specify the color to render the foreground of the meter when the value of the meter is lower than *LowAlarmLimit*. This color is also used to render the Low Alarm Limit Marker.

Property bool AlarmMarkerVisible

Specify the visibility of the markers to mark the alarm high and low limits.

Property bool AlarmValueVisible

Specify the visibility of the alarm high and low limits.

## Warning Properties:

The following properties are used to support warning display and warning event generation. These properties are similar to the alarm properties.

Property bool SupportWarning

Specify whether to support the warning function or not.

Property double HighWarningLimit

Specify the high limit that the value of the meter exceeds to be remarked as “High” warning.

Property double LowWarningLimit

Specify the low limit that the value of the meter exceeds to be remarked as “Low” warning.

Property Color HighWarningColor



Specify the color to render the foreground of the meter when the value of the meter is higher than *HighWarningLimit* (Alarm takes high priority than Warning for the foreground color). This color is also used to render the High Warning Limit Marker.

Property Color LowWarningColor

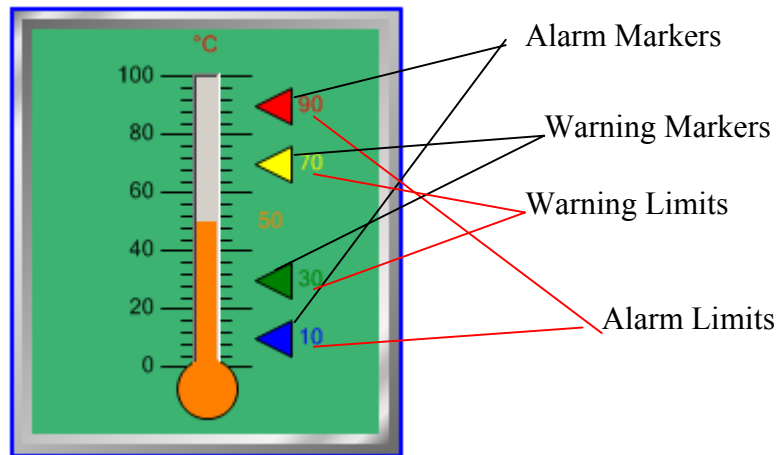
Specify the color to render the foreground of the meter when the value of the meter is lower than *LowWarningLimit* (Alarm takes high priority than Warning for the foreground color). This color is also used to render the Low Warning Limit Marker.

Property bool WarningMarkerVisible

Specify the visibility of the markers to mark the warning high and low limits.

Property bool WarningValueVisible

Specify the visibility of the warning high and low limits.



## Events:

First `AWEventArgs` is introduced. There are three arguments in `Object` sender,  
`AWEventArgs.Limit` //Alarm or Warning Limit  
`AWEventArgs.ActualValue` //Actual value of the meter  
`AWEventArgs.Active` //The alarm or warning is active or not.

There are four events can be generated when the corresponding functions are enabled (SupportAlarm and SupportWarning).

`HighAlarm(Object sender, AWEventArgs e)`

Triggered when the value of the meter changes from the value point which is lower than `HighAlarmLimit` to the value point which is equal to or higher than `HighAlarmLimit` (`e.Active = True`), or from the value point which is equal to



or higher than HighAlarmLimit to the value point which is lower than HighAlarmLimit (e.Active = False).

LowAlarm(Object sender, AWEventArgs e)

Triggered when the value of the meter changes from the value point which is higher than LowAlarmLimit to the value point which is equal to or lower than LowAlarmLimit (e.Active =True), or from the value point which is equal to or lower than LowAlarmLimit to the value point which is higher than LowAlarmLimit (e.Active = False).

HighWarning(Object sender, AWEventArgs e)

Triggered when the value of the meter changes from the value point which is lower than HighWarningLimit to the value point which is equal to or higher than HighWarningLimit (e.Active =True), or from the value point which is equal to or higher than HighWarningLimit to the value point which is lower than HighWarningLimit (e.Active = False).

LowWarning(Object sender, AWEventArgs e)

Triggered when the value of the meter changes from the value point which is higher than LowWarningLimit to the value point which is equal to or lower than LowWarningLimit (e.Active =True), or from the value point which is equal to or lower than LowWarningLimit to the value point which is higher than LowWarningLimit (e.Active = False).