



## Using ActiveX Ruler Control

### 1. Overview

The Ruler ActiveX control allows users to visual show some dynamic values on a scaled ruler. This control can be used in many applications such as Factory Instrumentation Readouts, Process Control User Interface, Motion Control Monitor and other monitor displays.

This ActiveX product is very powerful for designers to configure different kinds of process monitor displays which present visual scale effect. There are a rich group of the properties which enable people to design many kinds of displays to their requirements. All these properties will be addressed in the following sections.

### 2. Interfaces

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

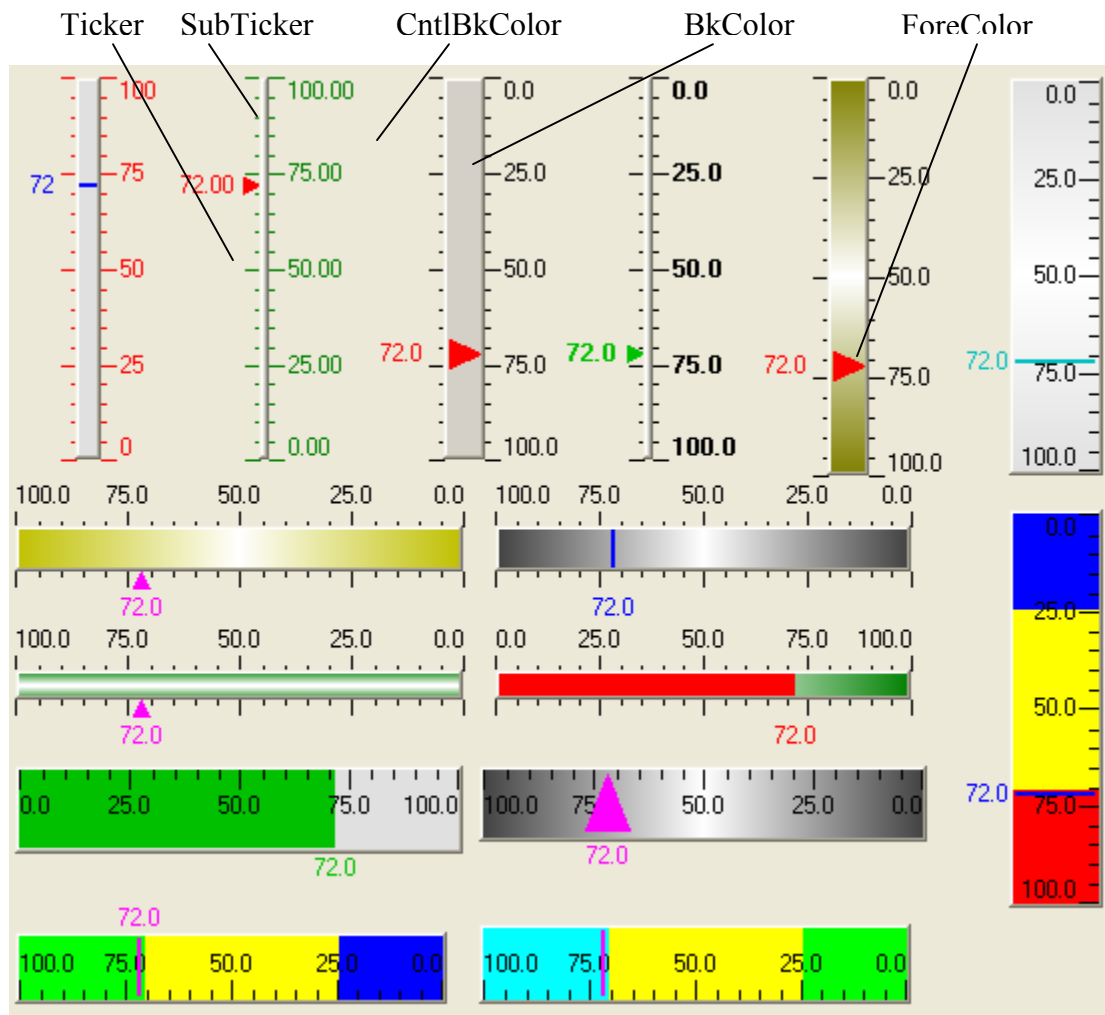


Figure 1



## Properties

### Property *BkColor* As *OLE\_COLOR*

*BkColor* specifies the color of the ruler bar.

### Property *BorderGap* As *Integer*

Specifies the gap between the border of the control and the ruler bar which gives adjustable room to display the scale values and actual value when they are very close to the control border.

### Property *CntlBkColor* As *OLE\_COLOR*

*CntlBkColor* specifies the background color of the control if “*BackStyle* = *BS\_Opaque*”.

### Property *BackStyle* As *BackStyleType*

If *BackStyle*=*BS\_Opaque*, then the background is rendered using *CntlBkColor*; If *BackStyle*=*BS\_Transparent*, then the background is rendered using the ambient background color of the container of the control.

### Property *DecimalNum* As *Integer*

Specifies the digit number of decimal part of the scale texts and the ruler value.

### Property *Enabled* As *Boolean*

Specifies whether the window of the ActiveX control is enabled or disabled. If disabled, there is no window message or event triggered by the control.

### Property *Font* As *IFontDisp*

Defines the standard font property to display the scale text, legend text and the value.

### Property *ForeColor* As *OLE\_COLOR*

*ForeColor* Specifies the color of the moving part (including actual value text and ruler indicator).

### Property *GradientType* As *GradientStyle*

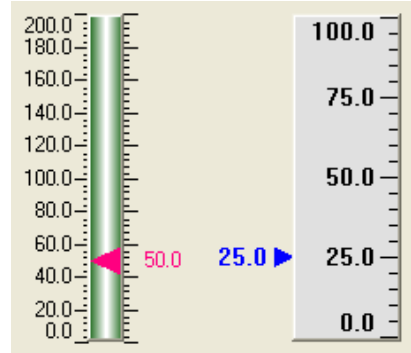
*GS\_NONE* = 1 : No gradient rendering in the ruler bar; *GS\_HORIZONTAL* = 2 : The background color of the ruler bar changes horizontal from *BkColor* to white and to *BkColor* again gradually; *GS\_VERTICAL* = 3 : The background color of the ruler bar changes vertically from *BkColor* to white and to *BkColor* again gradually

### Property *hWnd* As *Long*

Retrieves the window handler of the control. It's a read-only property.

### Property *InnerScale* As *Boolean*

Specifies whether the scale tickers (including sub tickers) and scale texts are shown inside the ruler bar or outside the ruler bar.



*InnerScale=FALSE InnerScale=TRUE*

Property InverseDirection As Boolean

*InverseDirection* specifies the Min⇒Max direction. If *InverseDirection=FALSE*, Min⇒Max direction is from left to right or from bottom to top; otherwise Min⇒Max direction is from right to left or from top to bottom.

Property MaxVal As Double

*MaxVal* specifies the maximum value of the ruler scale.

Property MinVal As Double

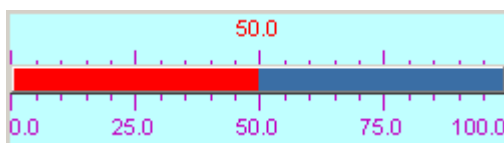
*MinVal* specifies the minimum value of the ruler scale.

Property ScaleSideChanged As Boolean

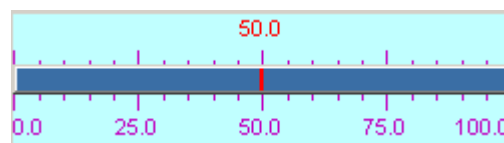
*ScaleSideChanged=TRUE* means the scale is drawn in the bottom side (if the ruler is horizontal) or right side (if the ruler is vertical). Otherwise the top side or the left side.

Property ShowValStyle As ShowValueStyle

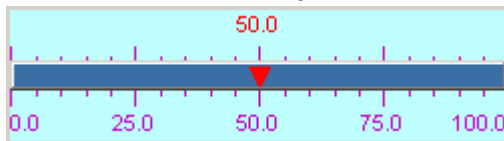
There are four types of the moving indicators, i.e.,



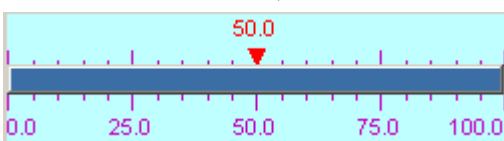
*BAR = 0*



*LINE = 1*



*INTRIANGLE = 2*



*OUTTRIANGLE = 3*

*NoIndicator = 4* means there is no actual value indicator. *ShowValStyle* specifies which actual value indicator style the ruler will choose.

Property TickerNum As Integer

Property SubTickerNum As Integer

*TickerNum* and *SubTickerNum* are used to define the scale ticker number and sub scale ticker number.



## Property TextColor As OLE\_COLOR

*TextColor* is used to specifies the scale text color and the scale ticker color.

## Property Value As Double

Sets or retrieves the value of the ruler.

## Property Vertical As Boolean

*Vertical* specifies the ruler bar is vertical or horizontal. If *Vertical* = *TRUE*, it's vertical, otherwise it's horizontal.

## Methods & Functions

### Sub SetScaleRangeColor(index As Integer, start As Double, end As Double, color As OLE\_COLOR)

### Sub CancelScaleRangeColor(index As Integer)

*SetScaleRangeColor* is used to configure the multi-color scale, where *index* defines a scale range identifier, *start* defines the range start value and *end* defines range end value, *color* specifies the range color of the specified scale range.

*CancelScaleRangeColor* is used to cancel the scale range specified by *index*. And the color of the range will be changed back to *BkColor*.

These two methods only work at *GradientType*= *GS\_NONE*.

### Sub UpdateValue(newVal As Double)

*UpdateValue* is used to update the value of the ruler to the new value specified by *newVal*.

## Events

### Event Click()

When the chart is clicked, the event is trigged to the container. Fired when the control captures the mouse, any **BUTTONUP** (left, middle, or right) message is received, and the button is released over the control. The **MouseDown** and **MouseUp** events occur before this event.

### Event DblClick()

When the chart is double clicked, the event is trigged to the container. Similar to **Click** but fired when a **BUTTONDBLCLK** message is received.

### Event KeyDown(nChar As Long, nRepCnt As Long, nFlags As Long)

Fired when a **WM\_SYSKEYDOWN** or **WM\_KEYDOWN** message is received.

*nChar* specifies the virtual key code of the given key. For a list of standard virtual key codes, see *Winuser.h* ; *nRepCnt* specifies the repeat count, that is, the number of times the keystroke is repeated as a result of the user holding down the key; *nFlags* specifies the scan code, key-transition code, previous key state, and context code. For details, please refer to MSDN.

### Event KeyUp(nChar As Long, nRepCnt As Long, nFlags As Long)

Fired when a **WM\_SYSKEYUP** or **WM\_KEYUP** message is received. Please refer to *KeyDown* event.

### Event MouseDown(x As Long, y As Long)



---

Fired if any **BUTTONDOWN** (left, middle, or right) is received. The mouse is captured immediately before this event is fired. x and y represent the corresponding coordinate values in the control, the origin is at the left-top point of the control.

Event MouseMove(x As Long, y As Long)

Fired when a **WM\_MOUSEMOVE** message is received.

Event MouseUp(x As Long, y As Long)

Fired if any **BUTTONUP** (left, middle, or right) is received. The mouse capture is released before this event is fired.