



Using ActiveX Indicator Control

1. Overview

There are so many situations which need to show the boolean discrete status, such as TRUE/FALSE, YES/NO, or ON/OFF indicators. The ActiveX Indicator controls are developed by Dragonfly Automation Software to visually show all the digital statuses. Different customers may need different vivid indicators to represent different digital statuses. For example, if there is no new email in the mailbox, then an empty mailbox icon will be shown on the screen, otherwise a mailbox with a flag will be shown to tell people that there are some new emails in the mailbox. In the industrial automation fields, there are so many switch variables, alarm variables, boolean process variables of which the statuses need to be shown on Machine Interface devices. Even at the web design, the indicators are very important for the whole page design, more vivid indicators can attract more attentions of the web users. Therefore for industrial applications or Web applications, we need more powerful indicator ActiveX control.

Requirements involving all these indicators vary based on different applications and variable types. To meet all these requirements, the ActiveX indicator controls provide many general default implementations to allow the users not to write any extra codes to get what they expect, meanwhile two properties, i.e., OnPicture and OffPicture, are designed to allow the customers to provide their own vivid images for ON and OFF status. The combination show of the image and text is also possible when you use this control. Powerful border style design is provided to make the customer design their own customized border style as they like. All these properties will be addressed in the following sections.

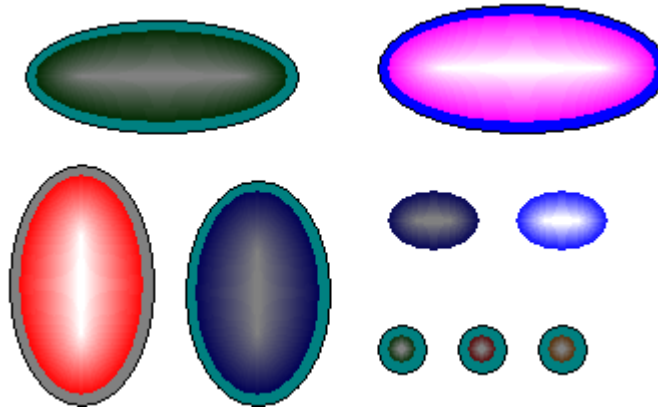
Matrix LED is a common indicator panel that can be used to display a number of discrete Boolean variables. A LED panel containing many indicators can be used to show the status of a group relative switch variables.

There are four indicator components provided in the Dragonfly ActiveX Indicator package, i.e., BallIndicator, BmpIndicator, ShapeIndicator and LEDMatrix. BallIndicator is designed as a visual LED; BmpIndicator is designed to allow the customer to provide their own images for ON/OFF statuses; ShapeIndicator can be configured to many different shapes to demonstrate the switch status; And LEDMatrix groups a number of indicators in a matrix panel. Its lightweight feature makes it a good option in the Web design. Its possible uses should include

- | | |
|--------------------------------|---|
| 1). Factory Instruments System | 2). Audible Warning or Danger Indicator |
| 3). ON/OFF Switch | 4). Blinking Attention Grabber |
| 5). Automatic Shutoff Control | 6). General Purpose Toggle or Switch |

2. BallIndicator

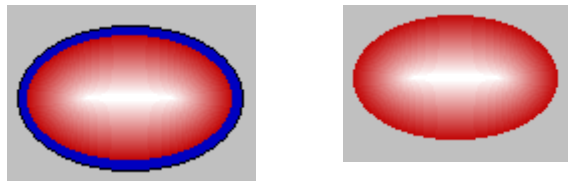
The designers can use the interface properties and methods of the BallIndicator component to implement some LED indicators (the shape can be circle or ellipse), the following picture shows some styles which can be configured.



Properties

Property *bCircleBorder* As Boolean

If *TRUE* is assigned to the property, then the indicator has one circle or ellipse border with the color specified by *BorderColor* property; If *False* is assigned to the property, then no border around the indicator.



Property *BkColor* As OLE_COLOR

Specifies the background color of the indicator if “*BackStyle* = *BS_Opaque*”.

Property *BackStyle* As BackStyleType

If *BackStyle*=*BS_Opaque*, then the background is rendered using *BkColor*; If *BackStyle*=*BS_Transparent*, then the background is rendered using the ambient background color of the container of the control.

Property *BorderColor* As OLE_COLOR

Specifies the border color of the circle or ellipse border around the indicator.

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 *hWnd* As Long

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

Property *OffColor* As OLE_COLOR

Specifies the indicator color when its value is *FALSE*.

Property *OnColor* As OLE_COLOR

Specifies the indicator color when its value is *TRUE*.



Property Value As Boolean

Specifies the Boolean status of the indicator.

Methods

Sub UpdateValue(newVal As Boolean)

This method is used to update the indicator status.

Events

Event Click()

When the chart is clicked, the event is triggered 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 triggered 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.

3. BmpIndicator

The designers can use the interface properties and methods of the BmpIndicator component to implement many kinds of image indicators, Some default ON/OFF images are provided by the



component which include LAMPIMG, LEDIMG, LOCKIMG and MAILIMG and other more than 18 default images. The following pictures show some styles which can be configured.



Properties:

Property *BkColor* As *OLE_COLOR*

BkColor is used to set the background color of control if “*BackStyle = BS_Opaque*”.

Property *BackStyle* As *BackStyleType*

If *BackStyle=BS_Opaque*, then the background is rendered using *BkColor*; If *BackStyle=BS_Transparent*, then the background is rendered using the ambient background color of the container of the control.

Property *BorderDarkColor* As *OLE_COLOR*

This property is used to configure the color of the border side which is hidden from the light. This property will be used in conjunction with *BorderLightColor* to create the 3D-Look border.



Property BorderLightColor As OLE_COLOR

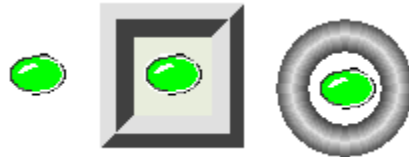
This property is used to configure the color of the border side which faces the light. This property will be used in conjunction with *BorderDarkColor* to create the 3D-Look border.

Property BorderType As Edge

There are three items in Edge enumeration definition,

- FLAT* = 0 (No Border)
- RECTANGLE* = 1 (Rectangle Border)
- CIRCLE* = 2 (Circle Border)

Which defines the Border style.

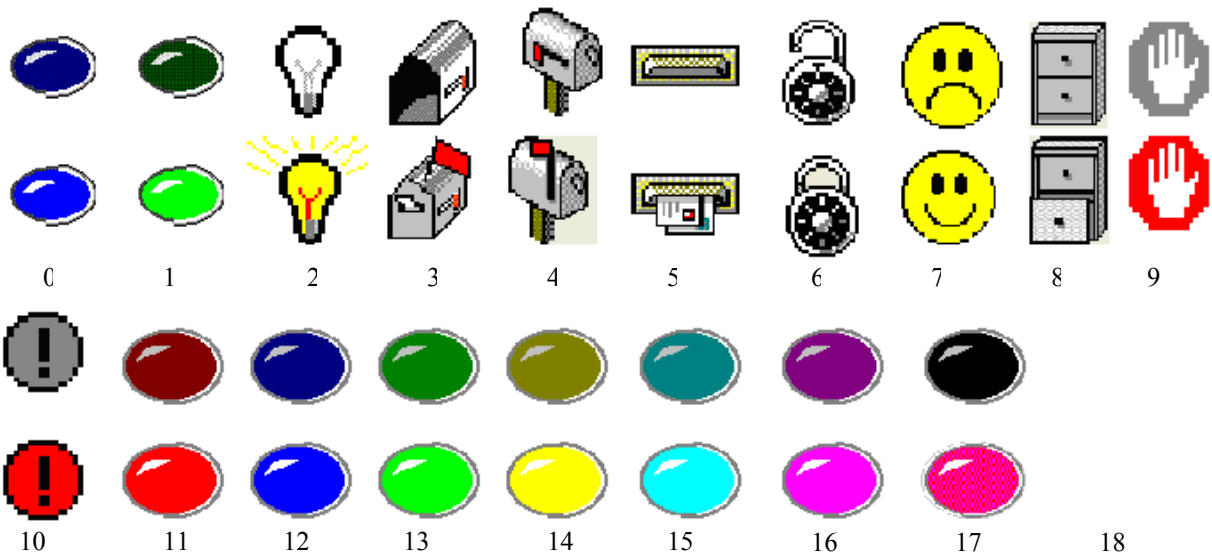


FLAT=0 RECTANGLE=1 CIRCLE=2

Property DefaultBitmap As BitmapDefault

BitmapDefault is an enumeration data type which define the following items,

- | | |
|----------------|----------------|
| LEDIMG =0 | LEDIMGGREEN =1 |
| LAMPIMG =2 | MAILIMG =3 |
| MAIL2IMG =4 | MAIL3IMG =5 |
| LOCKIMG =6 | FACEIMG =7 |
| FILEIMG =8 | ALARMIMG =9 |
| WARNINGIMG =10 | REDIMG =11 |
| BLUEIMG =12 | GREENIMG =13 |
| YELLOWIMG =14 | CYANIMG =15 |
| PINKIMG =16 | BLACKIMG =17 |
| OTHERIMG =18 | |





DefaultBitmap is used to specify the which default image is chosen as image of the control. *OTHERIMAGE(18)* is used to setup the image defined by *OffPicture* and *OnPicture* properties.

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 text.

Property hWnd As Long

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

Property InnerBorderLen As Integer

This property is used to specify the length of the inner border.

Property OffPicture As IPictureDisp

The user can apply this property to provide the image of the OFF status when *DefaultBitpmap=OTHERIMG*.

Property OffText As String

The user can use this property to provide the textual representation of the OFF status.

Property OffTextColor As OLE_COLOR

This property is used to specify the color of the *OffText*.

Property OnPicture As IPictureDisp

The user can apply this property to provide the image of the ON status when *DefaultBitpmap=OTHERIMG*.

Property OnText As String

The user can use this property to provide the textual representation of the ON status.

Property OnTextColor As OLE_COLOR

This property is used to specify the color of the *OnText*. The following is the one example of the combination of the texts and the images.



Property OriginalSize As Boolean

Specifies whether the image is drawn at its original size or automatically resize to the control size.

Property OuterBorderLen As Integer

This property is used to specify the length of the outer border.



Property Value As Boolean

Specify the Boolean status of the indicator.

Events

Event Click()

When the chart is clicked, the event is triggered 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 triggered 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*)

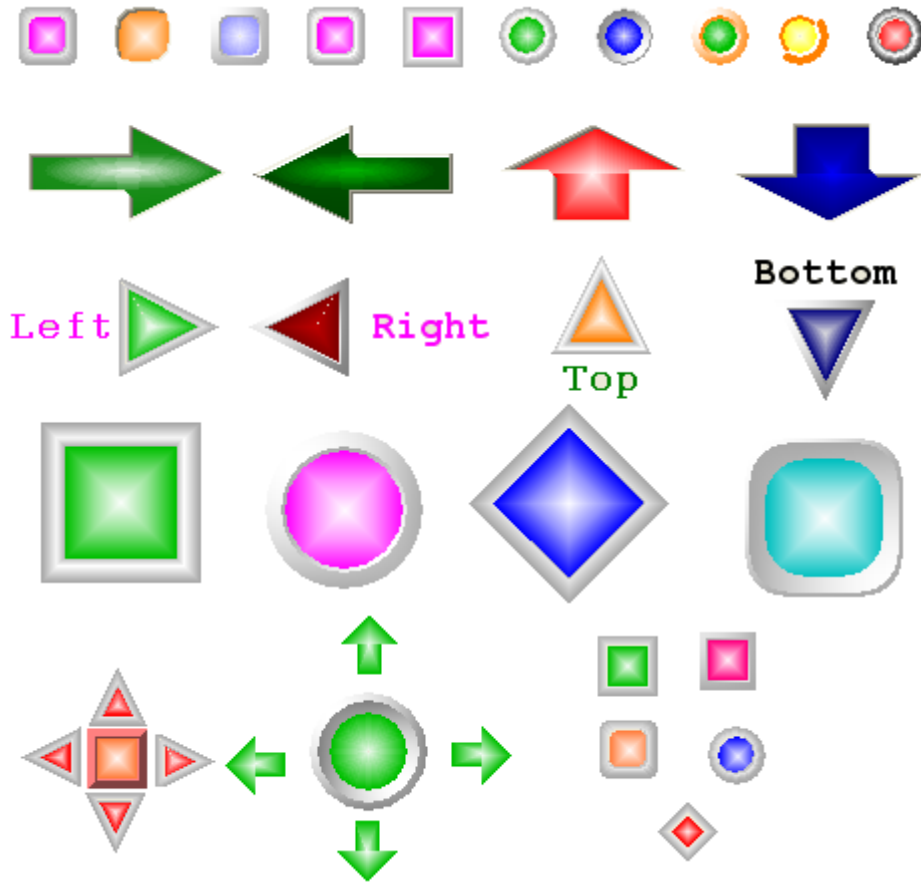
Fired when a **WM_MOUSEMOVE** message is received.

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.

3. ShapeIndicator

The **ShapeIndicator** control is another control which provides multiple indicator shapes for customers to configure different kinds of the discrete Boolean status display. The designers can use the interface properties and methods of the **ShapeIndicator** component to implement many kinds of indicators with different shapes as they like, the following pictures show some styles which can be configured.



Properties

Property *BkColor* As *OLE_COLOR*

BkColor is used to set the background color of the control if “*BackStyle = BS_Opaque*”.

Property *BackStyle* As *BackStyleType*

If *BackStyle*=*BS_Opaque*, then the background is rendered using *BkColor*; If *BackStyle* = *BS_Transparent*, then the background is rendered using the ambient background color of the container of the control.

Property *BorderDarkColor* As *OLE_COLOR*

This property is used to set the color of the border side which is hidden from the light. It is used in conjunction with *BorderLightColor* to create the 3D-Look border.

Property *BorderLightColor* As *OLE_COLOR*

This property is used to set the color of the border side which faces the light. it is used in conjunction with *BorderDarkColor* to create the 3D-Look border.

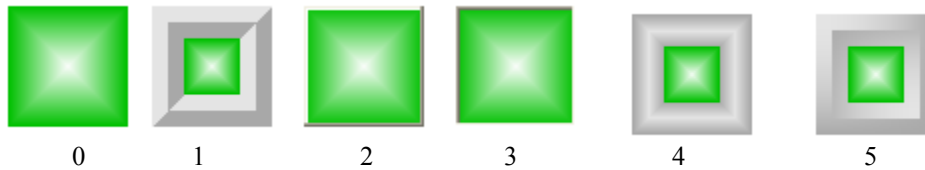
Property *BorderGradientType* As *BorderGradientStyle*

Six border gradient styles are defined as follows,

<i>BGS_NONE</i>	=0,	<i>BGS_SIMPLECONFIG</i>	=1,
<i>BGS_SIMPLERAISED</i>	=2,	<i>BGS_SIMPLESUNKEN</i>	=3,



BGS_LINEAR =4, *BGS_NONLINEAR* =5,
BorderGradientType is used to specify which border gradient style is used to render the border of the control.



Property Caption As String

Defines the caption of the control. If *TextVisible=TRUE*, the caption will be drawn on the indicator.

Property Enabled As Boolean

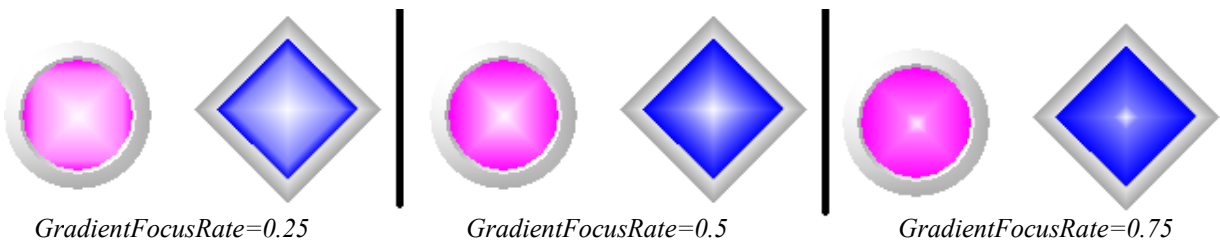
Specifies whether the window of the ActiveX control is enabled or disabled. If disabled, there is no window message or event triggered by the control.

Property Font As IFontDisp

Defines the standard font property to display the text.

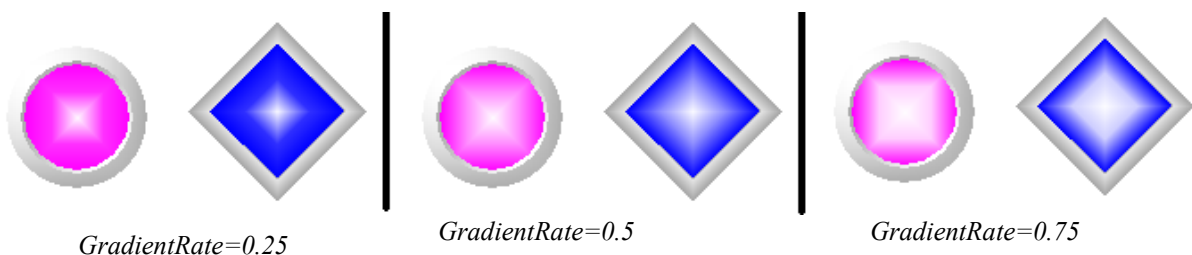
Property GradientFocusRate As Single

Specifies the focus rate in gradient rendering. Its range is from 0 to 1.0, 0 means the whole indicator rectangle is the focus rectangle (that implies no focus), the whole indicator is rendered by the gradient color; 1.0 means the whole indicator is rendered in the face color (OnColor or OffColor). 0⇒1.0 means that the focus rectangle changes from the rectangle of the indicator to the center point of the indicator.



Property GradientRate As Single

GradientRate changes the color change rate which results in the strength of the gradient color in the focus rectangle.



In the above figure, we see that the focus rectangles do not change, but the strength of the gradient colors change.



Property GradientType As GradientStyle

There are two definitions involving the gradient rendering, i.e.,

GS_CIRCLE=0,

GS_POLYGON=1



GS_CIRCLE supposes that the all points of the frame of the circle or ellipse should have color; GS_POLYGON supposes that all the points of the frame of the polygon (rectangle or triangle or other polygon shapes) should have the same color.

Property hWnd As Long

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

Property IndicatorHeight As Integer

Specifies the height of the indicator. Note that the indicator can not be automatically resized to the size of the control.

Property IndicatorWidth As Integer

Specifies the width of the indicator. Note that the indicator can not be automatically resized to the size of the control.

Property InnerBorderLen As Integer

This property is used to specify the length of the inner border.

Property OffColor As OLE_COLOR

This property is used to specify the face color of the indicator at *OFF* status.

Property OffGradientColor As OLE_COLOR

This property is used to specify the gradient color of the indicator at *OFF* status.

Property OnColor As OLE_COLOR

This property is used to specify the face color of the indicator at *ON* status.

Property OnGradientColor As OLE_COLOR

This property is used to specify the gradient color of the indicator at *ON* status.

Property OuterBorderLen As Integer

This property is used to specify the length of the outer border.

Property RoundRadius As Integer

Specifies the round radius of the indicator when the shape of the indicator is the round rectangle (*ShapeType= FSS_ROUND_RECT*).



Property ShapeType As FaceShapeStyle

This is the most important property of ShapeIndicator control. The following shapes are defined,

FSS_RECT	=0,	FSS_ROUND_RECT	=1,
FSS_DIAMOND	=2,	FSS_CIRCLE	=3,
FSS_LEFT_TRIANGLE	=4,	FSS_RIGHT_TRIANGLE	=5,
FSS_TOP_TRIANGLE	=6,	FSS_BOTTOM_TRIANGLE	=7,
FSS_LEFT_ARROW	=8,	FSS_RIGHT_ARROW	=9,
FSS_TOP_ARROW	=10,	FSS_BOTTOM_ARROW	=11,

It should be easy for people to find the corresponding indicators in the first Figure of this section for the above definitions.

Property TextColor As OLE_COLOR

Specifies the color of the caption.

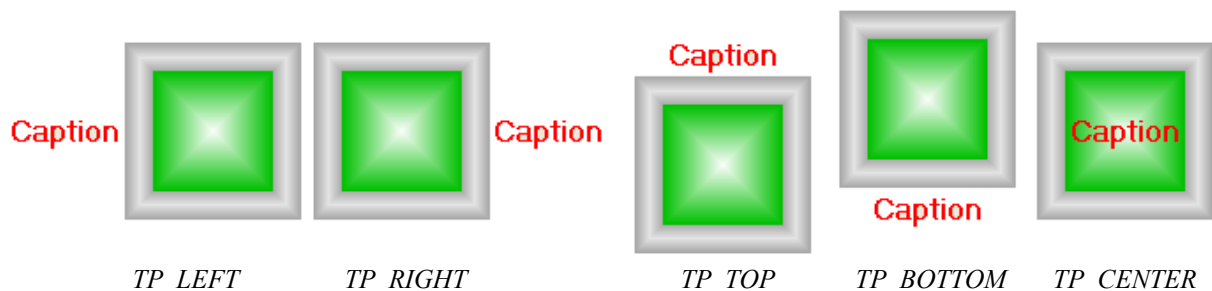
Property TextVisible As Boolean

Specifies whether the caption of the indicator is displayed or not.

Property TextPosition As TextPos

Specifies the caption position, there are five definitions of text position,

TP_LEFT	=0,	TP_RIGHT	=1,
TP_TOP	=2,	TP_BOTTOM	=3,
TP_CENTER	=4,		



Property TextAlignment As TextAlignmentStyle

Specifies how the caption text is aligned, there are two definitions,

TAS_TOINDICATOR=0,	// Text is aligned to the indicator side;
TAS_TOBORDER=1,	//Text is aligned to the border of the control.

Property Value As Boolean

Set or retrieve the Boolean status of the indicator.

Methods

Sub UpdateValue(newVal As Boolean)

This method is used to update the indicator status.



Events

Event Click()

When the chart is clicked, the event is triggered 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 triggered 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.

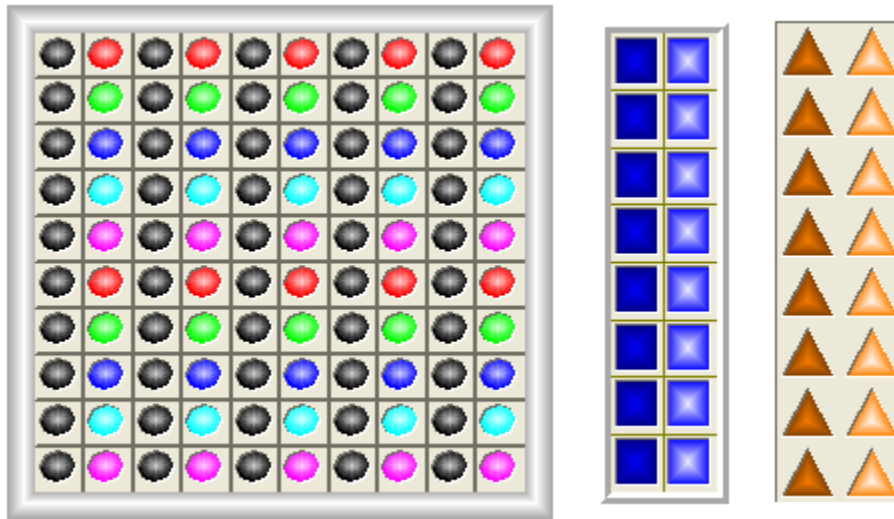
Event MouseUp(x As Long, y As Long)

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

4. LEDMatrix

The LEDMatrix control is a new control in this release as well which provides to the function that group all the indicators in one LED panel. The following pictures show some styles which can be configured.





Properties

Property *BkColor* As *OLE_COLOR*

BkColor is used to set the background color of the control if “*BackStyle* = *BS_Opaque*”.

Property *BackStyle* As *BackStyleType*

If *BackStyle*=*BS_Opaque*, then the background is rendered using *BkColor*; If *BackStyle* = *BS_Transparent*, then the background is rendered using the ambient background color of the container of the control.

Property *BorderDarkColor* As *OLE_COLOR*

This property is used to set the color of the border side which is hidden from the light. It is used in conjunction with *BorderLightColor* to create the 3D-Look border.

Property *BorderLightColor* As *OLE_COLOR*

This property is used to set the color of the border side which faces the light. It is used in conjunction with *BorderDarkColor* to create the 3D-Look border.

Property *BorderGradientType* As *BorderGradientStyle*

BorderGradientType is used to specify which border gradient style is used to render the border of the control. (Please refer to the same property of the *ShapeIndicator* control in last section.)

Property *ColumnNum* As *Integer*

Specifies the number of the columns of the LED panel matrix table.

Property *RowNum* As *Integer*

Specifies the number of the rows of the LED panel matrix table.



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 GridColor As OLE_COLOR

Specifies the color of the vertical grid lines between columns and the horizontal grid lines between rows of the LED Matrix table.

Property GridVisible As Boolean

Specifies whether the grid lines are visible or not.

Property GridWidth As Integer

Specifies the width of the grid lines.

Property hWnd As Long

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

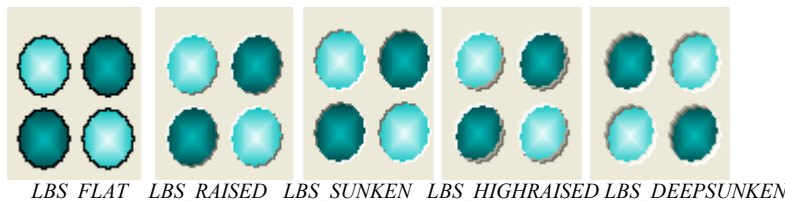
Property InnerBorderLen As Integer

This property is used to specify the length of the inner border of the control.

Property LEDBorderStyle As LEDBorderStyle

Specifies the border for each led indicator inside the panel.

<i>LBS_FLAT</i>	=0,	<i>LBS_RAISED</i>	=1,
<i>LBS_SUNKEN</i>	=2,	<i>LBS_HIGHRAISED</i>	=3,
<i>LBS_DEEPSUNKEN</i>	=4,		



Property LEDGap As Integer

Specifies the gap between the led indicators and the gridlines.

Property LEDGradientType As GradientStyle

Specifies the gradient rendering style. (Please refer to property *GradientType* of *ShapeIndicator* in the last section.)

Property LEDShape As FaceShapeStyle

Specifies which shape the LED indicator will take. Note only 0 —7 are supported in this LEDMatrix control, *FSS_LEFT_ARROW*(8), *FSS_RIGHT_ARROW*(9), *FSS_TOP_ARROW*(10) and *FSS_BOTTOM_ARROW*(11) are not supported. Please refer to Property *ShapeType* of the *ShapeIndicator* control.

Property OffColor As OLE_COLOR

This property is used to specify the default face color of the indicator at *OFF* status.



And the designer can change the *OffColor* of each individual indicator by calling *SetLEDProperties*

Property OffGradientColor As OLE_COLOR

This property is used to specify the gradient color of each indicator at *OFF* status.

Property OnColor As OLE_COLOR

This property is used to specify the default face color of the indicator at *ON* status. And the designer can change the *OnColor* of each individual indicator by calling *SetLEDProperties*

Property OnGradientColor As OLE_COLOR

This property is used to specify the gradient color of each indicator at *ON* status.

Property OuterBorderLen As Integer

This property is used to specify the length of the outer border of the control.

Property RoundRadius As Integer

Specifies the round radius of each indicator when the shape of the indicator is the round rectangle (*LEDShapeType= FSS_ROUND_RECT*).

Methods & Functions

Function GetLEDValue(iRow As Integer, iColumn As Integer, pVal As Boolean) As Integer

Retrieve the value of the LED indicator specified by the Row Index *iRow* and the Column Index *iColumn*. The returned value is saved to the *pVal*. If call is successful, then 0 is returned, otherwise 1 is returned.

Function SetLEDProperties(iRow As Integer, iColumn As Integer, OnColor As OLE_COLOR, OffColor As OLE_COLOR, UIUpdate As Boolean) As Integer

Uses this function to setup the properties of the LED indicator specified by the Row Index *iRow* and the Column Index *iColumn*. *OnColor* specifies the LED face color which its status is *ON* and *OffColor* specifies the LED face color which its status is *OFF*. *UIUpdate* specifies whether to refresh the control. If call is successful, then 0 is returned, otherwise 1 is returned.

Function UpdateLEDValue(iRow As Integer, iColumn As Integer, newVal As Boolean, UIUpdate As Boolean) As Integer

This method is used to update the status of the LED indicator specified by the Row Index *iRow* and the Column Index *iColumn* with *newVal*. *UIUpdate* specifies whether to refresh the control. If call is successful, then 0 is returned, otherwise 1 is returned.

Events

Event Click()



When the chart is clicked, the event is triggered 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 *DbClick()*

When the chart is double clicked, the event is triggered 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.