



ExCalc

Exontrol's ExCalc component provides calculator features to your application. The exCalc component provides implementation for arithmetic operations like addition, subtraction, division and multiplication.

Features include:

- **WYSWYG Template/Layout** Editor support
- Drop down version included
- Support for arithmetic operation like +, -, * or /
- Built-in **HTML** caption support
- Ability to specify the layout for the calculator's matrix
- Ability to specify the pictures for up or down buttons
- Picture background support
- Custom buttons support
- Ability to execute operations without showing the control's drop down portion
- and more



Ž ExCalc is a trademark of Exontrol. All Rights Reserved.

How to get support?

To keep your business applications running, you need support you can count on.

Here are few hints what to do when you're stuck on your programming:

- Check out the samples - they are here to provide some quick info on how things should be done
- Check out the how-to questions using the [eXHelper](#) tool
- Check out the help - includes documentation for each method, property or event
- Check out if you have the latest version, and if you don't have it send an update request [here](#).
- Submit your problem(question) [here](#).

Don't forget that you can contact our development team if you have ideas or requests for new components, by sending us an e-mail at support@exontrol.com (please include the name of the product in the subject, ex: exgrid) . We're sure our team of developers will try to find a way to make you happy - and us too, since we helped.

Regards,
Exontrol Development Team

<https://www.exontrol.com>

constants AppearanceEnum

Specifies the source's appearance.

Name	Value	Description
None2	0	The source has no borders.
Flat	1	Flat border
Sunken	2	Sunken border
Raised	3	Raised border
Etched	4	Etched border
Bump	5	Bump border

constants MessageEnum

The MessageEnum type specifies the list of values that can be changed using the [Message](#) property.

Name	Value	Description
exCannotDivideByZero	0	Specifies the message "Cannot divide by zero."

constants **PictureDisplayEnum**

Specifies how a picture object is displayed. Use the [PictureDisplay](#) property to align the control's [Picture](#) on its background.

Name	Value	Description
UpperLeft	0	Aligns the picture to the upper left corner.
UpperCenter	1	Centers the picture on the upper edge.
UpperRight	2	Aligns the picture to the upper right corner.
MiddleLeft	16	Aligns horizontally the picture on the left side, and centers the picture vertically.
MiddleCenter	17	Puts the picture on the center of the source.
MiddleRight	18	Aligns horizontally the picture on the right side, and centers the picture vertically.
LowerLeft	32	Aligns the picture to the lower left corner.
LowerCenter	33	Centers the picture on the lower edge.
LowerRight	34	Aligns the picture to the lower right corner.
Tile	48	Tiles the picture on the source.
Stretch	49	The picture is resized to fit the source.

Calc object

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {74B7322B-D54B-47AD-A891-AC60B02EE192}. The object's program identifier is: "Exontrol.Calc". The /COM object module is: "ExCalc.dll"

The Exontrol's Calc component provides calculator features to your application. The Calc object supports the following properties and methods:

Name	Description
Appearance	Retrieves or sets the control's appearance.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
BackColor	Specifies the control's background color.
BeginUpdate	Maintains performance when items are added to the control one at a time. This method prevents the control from painting until the EndUpdate method is called.
ButtonFromPoint	Retrieves the button from the point.
ButtonHeight	Specifies the height of the control's buttons.
Buttons	Specifies the list of buttons in the control.
ButtonWidth	Specifies the width of the control's buttons.
CalcHeight	Gets the height in pixels of the painted area.
CalcWidth	Gets the width in pixels of the painted area.
Caption	Specifies the control's caption.
Copy	Copies the control's content to the clipboard.
DecimalSymbol	Specifies the current decimal symbol.
EditBackColor	Specifies the control's edit background color.
EditForeColor	Specifies the control's edit foreground color.
EditHeight	Specifies the height of the control's edit portion.
Enabled	Enables or disables the control.
EndUpdate	Resumes painting the control after painting is suspended by the BeginUpdate method.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
Execute	Executes a command.
ExecuteTemplate	Executes a template and returns the result.

Font	Retrieves or sets the control's font.
ForeColor	Specifies the control's foreground color.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the control's window handle.
Images	Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.
ImageSize	Retrieves or sets the size of icons the control displays..
Message	Retrieves or sets a value that indicates the control's message.
Paste	Inserts data from the clipboard.
Picture	Retrieves or sets a graphic to be displayed in the control.
PictureDisplay	Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background
PictureDown	Specifies the picture that's displayed when the button is down.
PictureUp	Specifies the picture that's displayed when the button is up.
Refresh	Refreshes the control.
Replacelcon	Adds a new icon, replaces an icon or clears the control's image list.
Reset	Resets the control
ShowImageList	Specifies whether the control's image list window is visible or hidden.
Template	Specifies the control's template.
TemplateDef	Defines inside variables for the next Template/ExecuteTemplate call.
TemplatePut	Defines inside variables for the next Template/ExecuteTemplate call.
ToolTipDelay	Specifies the time in ms that passes before the ToolTip appears.
ToolTipPopDelay	Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.
Version	Retrieves the control's version.

property Calc.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the control's border style.

Use the Appearance property to define the control's border style. Use the Appearance property to hide the control borders.

method Calc.AttachTemplate (Template as Variant)

Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.

Type	Description
Template as Variant	A string expression that specifies the Template to execute.

The AttachTemplate/x-script code is a simple way of calling control/object's properties, methods/events using strings. The AttachTemplate features allows you to attach a x-script code to the component. The AttachTemplate method executes x-script code (including events), from a string, file or a safe array of bytes. This feature allows you to run any x-script code for any configuration of the component /COM, /NET or /WPF. Exontrol owns the x-script implementation in its easiest form and it does not require any VB engine or whatever to get executed. The x-script code can be converted to several programming languages using the eXHelper tool.

The following sample opens the Windows Internet Explorer once the user clicks the control (/COM version):

```
AttachTemplate("handle Click(){ CreateObject(`internetexplorer.application`){ Visible = True; Navigate(`https://www.exontrol.com`) } } ")
```

This script is equivalent with the following VB code:

```
Private Sub Calc1_Click()  
    With CreateObject("internetexplorer.application")  
        .Visible = True  
        .Navigate ("https://www.exontrol.com")  
    End With  
End Sub
```

The AttachTemplate/x-script syntax in BNF notation is defined like follows:

```
<x-script> := <lines>  
<lines> := <line>[<eol> <lines>] | <block>  
<block> := <call> [<eol>] { [<eol>] <lines> [<eol>] } [<eol>]  
<eol> := ";" | "\r\n"  
<line> := <dim> | <createobject> | <call> | <set> | <comment> | <handle>[<eol>][<eol>]  
<lines>[<eol>][<eol>]  
<dim> := "DIM" <variables>  
<variables> := <variable> [, <variables>]
```

```

<variable> := "ME" | <identifier>
<createobject> := "CREATEOBJECT(`"<type>`")"
<call> := <variable> | <property> | <variable>."<property>" | <createobject>."<property>"
<property> := [<property>"."]<identifier>["("<parameters>")"]
<set> := <call> "=" <value>
<property> := <identifier> | <identifier> "(" [<parameters>] ")"
<parameters> := <value> [","<parameters>]
<value> := <boolean> | <number> | <color> | <date> | <string> | <createobject> | <call>
<boolean> := "TRUE" | "FALSE"
<number> := "0X"<hexa> | ["-"]<integer>["."<integer>]
<digit10> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit16> := <digit10> | A | B | C | D | E | F
<integer> := <digit10> [<integer>]
<hexa> := <digit16> [<hexa>]
<color> := "RGB("<integer>","<integer>","<integer>")"
<date> := "#"<integer>"/"<integer>"/"<integer>" "["<integer>":"<integer>":"<integer>"]"#
<string> := ""<text>"" | ""<text>""
<comment> := ""<text>
<handle> := "handle " <event>
<event> := <identifier> "(" [<eparameters>] ")"
<eparameters> := <eparameter> [","<eparameters>]
<parameters> := <identifier>

```

where:

<identifier> indicates an identifier of the variable, property, method or event, and should start with a letter.

<type> indicates the type the CreateObject function creates, as a progID for /COM version or the assembly-qualified name of the type to create for /NET or /WPF version

<text> any string of characters

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character.

The advantage of the AttachTemplate relative to [Template](#) / [ExecuteTemplate](#) is that the AttachTemplate can add handlers to the control events.

property Calc.BackColor as Color

Specifies the control's background color.

Type	Description
Color	A color expression that indicates the control's background color.

Use the BackColor property to set the control's background color. Use the [ForeColor](#) property to change the control's foreground color. Use the [EditBackColor](#) property to specify the control's edit background color. Use the [EditForeColor](#) property to specify the control's edit foreground color.

method Calc.BeginUpdate ()

Maintains performance when items are added to the control one at a time.

Type	Description
------	-------------

This method prevents the control from painting until the EndUpdate method is called.

property Calc.ButtonFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as String

Retrieves the button from the point.

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates
String	A string expression that indicates the name of the button over the point (x,y) excluding the HTML tags.

Use the ButtonFromPoint property to get the button from cursor.

The following sample displays the button from cursor:

```
Private Sub Calc1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With Calc1
    Debug.Print .ButtonFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
  End With
End Sub
```

property Calc.ButtonHeight as Long

Specifies the height of the control's buttons.

Type	Description
Long	A long expression that indicates the height of the buttons, in pixels.

By default, the ButtonHeight property is 24 pixels. Use the ButtonHeight and [ButtonWidth](#) property to specify the height and the width of the buttons in the control. Use the [PictureDown](#) and [PictureUp](#) properties to specify the picture for the buttons. Use the [Buttons](#) property to customize the control's matrix of buttons. Use the [EditHeight](#) property to specify the height of the control's edit portion.

property Calc.Buttons as String

Specifies the list of buttons in the control.

Type	Description
String	A string expression that indicates the list of buttons being displayed. The rows are separated by chr(13)+chr(10) (vbCrLf) sequence, and the buttons inside the row are separated by ';' character.

The Buttons property specifies the buttons and the layout of the buttons in the control. Use the [PictureDown](#) and [PictureUp](#) properties to specify the picture for the buttons. Use the [ButtonHeight](#) and [ButtonWidth](#) property to specify the height and the width of the buttons in the control. The [ClickButton](#) event occurs when the user clicks a button in the calculator. You can use the [ButtonFromPoint](#) property to get the button from the point. The [DecimalSymbol](#) property returns the current decimal symbol. The system's decimal symbol can be changed in your Control panel, under the Regional settings.

Each button supports built-in HTML format like follows:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... ** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "**bit**" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "**bit**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline>

... `</solidline>` draws a black solid-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.

- **`<dotline rrggbb> ... </dotline>` or `<dotline=rrggb> ... </dotline>`** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The `<dotline> ... </dotline>` draws a black dot-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.
- **`<upline> ... </upline>`** draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).
- **`<r>`** right aligns the text
- **`<c>`** centers the text
- **`
`** forces a line-break
- **`number[:width]`** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **`key[:width]`** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **`&`** glyph characters as **`&`**; (`&`), **`<`**; (`<`), **`>`**; (`>`), **`&qout;`**; (`"`) and **`&#number;`**; (the character with specified code), For instance, the `€` displays the EUR character. The `&` ampersand is only recognized as markup when it is followed by a known letter or a `#` character and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`;
- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `` to define a smaller or a larger font to be displayed. For instance: "Text with `<off 6>`subscript" displays the text such as: Text with subscript The "Text with `<off -6>`superscript" displays the text such as: Text with superscript
- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the `rr/gg/bb` represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `` HTML tag can be used to define the height of the font. Any of the `rrggb`, mode or blend field may not be specified. The `<gra>` with no fields, shows a vertical gradient

color from the current text color to gray (808080). For instance the "<gradient-center" generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<out 000000><fgcolor=FFFFFF>outlined</fgcolor></out>" generates the following picture:

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<sha>shadow</sha>" generates the following picture:

shadow

or "<sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha>" gets:

outline anti-aliasing

By default, the Buttons property is "<fgcolor=0000FF>7</fgcolor>;<fgcolor=0000FF>8</fgcolor>;<fgcolor=0000FF>9</fgcolor>;<fgcolor=FF0000>/</fgcolor>;C\r\n<fgcolor=0000FF>4</fgcolor>;<fgcolor=0000FF>5</fgcolor>;<fgcolor=0000FF>6</fgcolor>;<fgcolor=FF0000>*</fgcolor>;1/x\r\n<fgcolor=0000FF>1</fgcolor>;<fgcolor=0000FF>2</fgcolor>;<fgcolor=0000FF>3</fgcolor>;<fgcolor=FF0000>-</fgcolor>;sqrt\r\n<fgcolor=0000FF>0</fgcolor>;<fgcolor=0000FF>+/-</fgcolor>;<fgcolor=0000FF>.</fgcolor>;<fgcolor=FF0000>+</fgcolor>;<fgcolor=000080>=</fgcolor>"

Use the [Caption](#) property to access the control's caption. Use the [Execute](#) method to execute operations inside the control.

The following sample adds a new button 'sin' and execute the trigonometric sin function when 'sin' button is clicked:

```
Private Sub Form_Load()  
    With Calc1  
        .Buttons = .Buttons + vbCrLf + "<b>sin<b>"  
    End With  
End Sub
```

```
Private Sub Calc1_ClickButton(ByVal Button As String, Cancel As Variant)  
    If (Button = "sin") Then  
        With Calc1  
            .Execute Sin(.Caption)  
        End With  
    End If  
End Sub
```

property Calc.ButtonWidth as Long

Specifies the width of the control's buttons.

Type	Description
Long	A long expression that indicates the width of the buttons, in pixels.

By default, the ButtonWidth property is 32 pixels. Use the [ButtonHeight](#) and ButtonWidth property to specify the height and the width of the buttons in the control. Use the [PictureDown](#) and [PictureUp](#) properties to specify the picture for the buttons. Use the [Buttons](#) property to customize the control's matrix of buttons.

property Calc.CalcHeight as Long

Gets the height in pixels of the painted area.

Type	Description
Long	A long expression that indicates the height in pixels of the control to fit all the buttons in the control's client area.

Use the CalcHeight and [CalcWidth](#) properties to get the size of the control to fit all the buttons inside the control's client area. Use the [Buttons](#) property to customize the control's matrix of buttons. Use the [ButtonHeight](#) and [ButtonWidth](#) property to specify the height and the width of the buttons in the control. The CalcWidth and CalcHeight properties does not include the size of the borders. Use the [Appearance](#) property to remove the control's borders.

The following sample resizes the control to fit all the buttons in the control's visible area:

```
Private Sub Form_Resize()  
    With Calc1  
        .Width = .CalcWidth * Screen.TwipsPerPixelX  
        .Height = .CalcHeight * Screen.TwipsPerPixelY  
    End With  
End Sub
```

property Calc.CalcWidth as Long

Gets the width in pixels of the painted area.

Type	Description
Long	A long expression that indicates the height in pixels of the control to fit all the buttons in the control's client area.

Use the [CalcHeight](#) and CalcWidth properties to get the size of the control to fit all the buttons inside the control's client area. Use the [Buttons](#) property to customize the control's matrix of buttons. Use the [ButtonHeight](#) and [ButtonWidth](#) property to specify the height and the width of the buttons in the control. The CalcWidth and CalcHeight properties does not include the size of the borders. Use the [Appearance](#) property to remove the control's borders.

The following sample resizes the control to fit all the buttons in the control's visible area:

```
Private Sub Form_Resize()  
    With Calc1  
        .Width = .CalcWidth * Screen.TwipsPerPixelX  
        .Height = .CalcHeight * Screen.TwipsPerPixelY  
    End With  
End Sub
```

property Calc.Caption as String

Specifies the control's caption.

Type	Description
String	A string expression that specifies the control's caption.

Use the Caption property to access the control's caption. The [Change](#) event notifies your application when the control's caption is changed. The Caption property erases the control's operators and operation stack and replaces the control's caption. By default, the Caption property replaces the control's stack (operators and operations) with giving caption. If the new caption just change the format of the Caption property (includes just HTML tags), the new format is applied to the control's label and control's stack (operators and operations) is not altered. For instance, let's say that the control's label displays the number 78, and during the [Change](#) event your application change the Caption property to "<sha ;;0>" + Caption. In this case, the content of the Caption is not change, instead just the new HTML format is applied to the control's label, so operations on the calculator can continue. Use the [Execute](#) method to execute operations. Use the [Buttons](#) property to assign a different matrix of digits and operators to the control. Use the [Reset](#) method to reset the control. The Reset method erases the control's caption as well as the internal stack of operators and operations.

The following sample displays the control's caption as soon as user types characters in the control:

```
Private Sub Calc1_Change()  
    With Calc1  
        Debug.Print .Caption  
    End With  
End Sub
```

The Caption property supports built-in HTML format like follows:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... ** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "**bit**" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "**bit**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The **<solidline> ... </solidline>** draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The **<dotline> ... </dotline>** draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires **<solidline>** or **<dotline>**).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **number[:width]** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **key[:width]** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&**; (&), **<**; (<), **>**; (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the **€** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <off 6>subscript" displays the text such as: Text with subscript The "Text with <off -6>superscript" displays the text such as: Text with subscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<gra FFFFFFFF;1;1>gradient-center</gra>" generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<out 000000><fgcolor=FFFFFF>outlined</fgcolor></out>" generates the following picture:

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<sha>shadow</sha>" generates the following picture:

shadow

or "<sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha>" gets:

outline anti-aliasing

The following sample displays the 'Cannot **execute** the operation' string:

Calc1.Caption = "Cannot execute the operation."

method Calc.Copy ()

Copies the control's content to the clipboard.

Type	Description
------	-------------

The Copy method copies the control's caption to the clipboard. Use the [Caption](#) property to access the control's caption. Use the [Paste](#) method to paste the control's clipboard in the control's label area.

property Calc.DecimalSymbol as String

Specifies the current decimal symbol.

Type	Description
String	A String expression that indicates the current decimal symbol.

The DecimalSymbol property returns the current decimal symbol. The system's decimal symbol can be changed in your Control panel, under the Regional settings. Use the [Refresh](#) method to reinitialize the [Buttons](#) property to re-read the decimal symbol.

property Calc.EditBackColor as Color

Specifies the control's edit background color.

Type	Description
Color	A color expression that indicates the control's edit background color.

Use the EditBackColor property to specify the control's edit background color. Use the [EditForeColor](#) property to specify the control's edit foreground color. Use the [ForeColor](#) property to change the control's foreground color. Use the [BackColor](#) property to set the control's background color.

property Calc.EditForeColor as Color

Specifies the control's edit foreground color.

Type	Description
Color	A color expression that indicates the control's edit foreground color.

Use the EditForeColor property to specify the control's edit foreground color. Use the [EditBackColor](#) property to specify the control's edit background color. Use the [ForeColor](#) property to change the control's foreground color. Use the [BackColor](#) property to set the control's background color.

property Calc.EditHeight as Long

Specifies the height of the control's edit portion.

Type	Description
Long	A long expression that indicates the height of the control's edit in pixels.

By default, the EditHeight property is 24 pixels. Use the EditHeight property to specify the height of the control's edit portion. Use the EditHeight property on 0 to hide the control's edit.

property Calc.Enabled as Boolean

Enables or disables the control.

Type	Description
Boolean	A boolean expression that indicates whether the control is enabled or disabled.

Use the Enabled property to enable or disable the control.

method Calc.EndUpdate ()

Resumes painting the control after painting is suspended by the BeginUpdate method.

Type	Description
------	-------------

property Calc.EventParam(Parameter as Long) as Variant

Retrieves or sets a value that indicates the current's event parameter.

Type	Description
Parameter as Long	A long expression that indicates the index of the parameter being requested ie 0 means the first parameter, 1 means the second, and so on. If -1 is used the EventParam property retrieves the number of parameters. Accessing an not-existing parameter produces an OLE error, such as invalid pointer (E_POINTER)
Variant	A VARIANT expression that specifies the parameter's value.

The EventParam method is provided to allow changing the event's parameters passed by reference, even if your environment does not support changing it (uniPaas 1.5 (formerly known as eDeveloper), DBase, and so on). For instance, Unipaas event-handling logic cannot update ActiveX control variables by updating the received arguments. The EventParam(0) retrieves the value of the first parameter of the event, while the EventParam(1) = 0, changes the value of the second parameter to 0 (the operation is successfully, only if the parameter is passed by reference). The EventParam(-1) retrieves the number of the parameters of the current event.

Let's take the event "event KeyDown (**KeyCode** as Integer, ByVal Shift as Integer)", where the KeyCode parameter is passed by reference. For instance, put the KeyCode parameter on 0, and the arrow keys are disabled while the control has the focus.

In most languages you will type something like:

```
Private Sub Control1_KeyDown(KeyCode As Integer, Shift As Integer)
    KeyCode = 0
End Sub
```

In case your environment does not support events with parameters by reference, you can use a code like follows:

```
Private Sub Control1_KeyDown(KeyCode As Integer, Shift As Integer)
    Control1.EventParam(0) = 0
End Sub
```

In other words, the EventParam property provides the parameters of the current event for reading or writing access, even if your environment does not allow changing parameters by

reference.

Calling the EventParam property outside of an event produces an OLE error, such as pointer invalid, as its scope was designed to be used only during events.

method Calc.Execute (Command as String)

Executes a command.

Type	Description
Command as String	A string expression that indicates the newly control's caption or the operation being executed.

Use the Execute method to execute commands in the control. The Execute method adds operations and operators to the control's stack. The Execute method does not clear the control's stack. The [Caption](#) property erases the control's operators and operation stack and replaces the control's caption. The [Change](#) event notifies your application when the control's caption is changed.

The following sample multiplies two numbers:

```
With Calc1
    .Execute "12.123"
    .Execute "*"
    .Execute "-123"
    .Execute "="
End With
```

method Calc.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed
Return	Description
Variant	A String expression that indicates the result after executing the Template.

Use the ExecuteTemplate property to returns the result of executing a template file. Use the [Template](#) property to execute a template without returning any result. Use the ExecuteTemplate property to execute code by passing instructions as a string (template string).

For instance, the following sample retrieves the beginning date (as string) for the default bar in the first visible item:

```
Debug.Print Calc1.ExecuteTemplate("Items.ItemBar(FirstVisibleItem(),``,1)")
```

Most of our UI components provide a Template page that's accessible in design mode. No matter what programming language you are using, you can have a quick view of the component's features using the WYSWYG Template editor.

- Place the control to your form or dialog.
- Locate the Properties item, in the control's context menu, in design mode. If your environment doesn't provide a Properties item in the control's context menu, please try to locate in the Properties browser.
- Click it, and locate the Template page.
- Click the Help button. In the left side, you will see the component, in the right side, you will see a x-script code that calls methods and properties of the control.

The control's Template page helps user to initialize the control's look and feel in design mode, using the x-script language that's easy and powerful. The Template page displays the control on the left side of the page. On the right side of the Template page, a simple editor is displayed where user writes the initialization code. The control's look and feel is automatically updated as soon as the user types new instructions. The Template script is saved to the container persistence (when Apply button is pressed), and it is executed when the control is initialized at runtime. Any component that provides a WYSWYG Template page, provides a Template property. The Template property executes code from a string (template string).

The Template script is composed by lines of instructions. Instructions are separated by

"\n\r" (newline) characters.

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable = property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property(list of arguments) = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method(list of arguments) *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- { *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- } *Ending the object's context*
- object. property(list of arguments).property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property Calc.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object being used.

Use the Font property to assign a font to the control.

property Calc.ForeColor as Color

Specifies the control's foreground color.

Type	Description
Color	A color expression that indicates the control's foreground color.

Use the ForeColor property to change the control's foreground color. Use the [BackColor](#) property to set the control's background color. Use the [EditBackColor](#) property to specify the control's edit background color. Use the [EditForeColor](#) property to specify the control's edit foreground color.

property Calc.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	<p>The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:</p> <ul style="list-style-type: none">• a string expression that indicates the path to the picture file, being loaded.• a string expression that indicates the base64 encoded string that holds a picture object, Use the eximages tool to save your picture as base64 encoded format.• A Picture object that indicates the picture being added or replaced. (A Picture object implements IPicture interface), <p>If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added</p>

The HTMLPicture property handles a collection of custom size picture being displayed in the HTML captions, using the tags. By default, the HTMLPicture collection is empty. Use the HTMLPicture property to add new pictures to be used in HTML captions. For instance, the HTMLPicture("pic1") = "c:\winnt\zapotec.bmp", loads the zapotec picture and associates the pic1 key to it. Any "pic1" sequence in HTML captions, displays the pic1 picture. On return, the HTMLPicture property retrieves a Picture object (this implements the IPictureDisp interface).

The following sample shows how to put a custom size picture in the column's header:

```
<CONTROL>.HTMLPicture("pic1") = "c:/temp/editors.gif"
<CONTROL>.HTMLPicture("pic2") = "c:/temp/editpaste.gif"

<COLUMN1>.HTMLCaption = "A <img>pic1</img>"
<COLUMN2>.HTMLCaption = "B <img>pic2</img>"
<COLUMN3>.HTMLCaption = "A <img>pic1</img> + B <img>pic2</img>"
```


property Calc.hWnd as Long

Retrieves the control's window handle.

Type	Description
Long	A long value that indicates the handle of the control's window.

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

method Calc.Images (Handle as Variant)

Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.

Type	Description
------	-------------

The Handle parameter can be:

- A string expression that specifies the ICO file to add. The ICO file format is an image file format for computer icons in Microsoft Windows. ICO files contain one or more small images at multiple sizes and color depths, such that they may be scaled appropriately. For instance, Images("c:\temp\copy.ico") method adds the sync.ico file to the control's Images collection (*string, loads the icon using its path*)
- A string expression that indicates the BASE64 encoded string that holds the icons list. Use the Exontrol's [ExImages](#) tool to save/load your icons as BASE64 encoded format. In this case the string may begin with "gBJJ..." (*string, loads icons using base64 encoded string*)
- A reference to a Microsoft ImageList control (mscomctl.ocx, MSComctlLib.ImageList type) that holds the icons to add (*object, loads icons from a Microsoft ImageList control*)
- A reference to a Picture (IPictureDisp implementation) that holds the icon to add. For instance, the VB's LoadPicture (Function LoadPicture([FileName], [Size], [ColorDepth], [X], [Y]) As IPictureDisp) or LoadResPicture (Function LoadResPicture(id, restype As Integer) As IPictureDisp) returns a picture object (*object, loads icon from a Picture object*)
- A long expression that identifies a handle to an Image List Control (the Handle should be of HIMAGELIST type). On 64-bit platforms, the Handle parameter must be a Variant of LongLong / LONG_PTR data type (signed 64-bit (8-byte) integers), saved under lVal field, as VT_I8 type. The LONGLONG / LONG_PTR is __int64, a 64-bit integer. For instance, in C++ you can use as Images(COleVariant((LONG_PTR)hImageList)) or Images(COleVariant(

Handle as Variant

(LONGLONG)hImageList)), where hImageList is of HIMAGELIST type. The GetSafeHandle() method of the CImageList gets the HIMAGELIST handle (long, loads icon from HIMAGELIST type)

The control provides an image list window, that's displayed at design time. Use the [ShowImageList](#) property to hide the image list window, at design time. At design time, the user can add new icons to the control's Images collection, by dragging icon files, exe files, etc, to the images list window. At runtime, the user can use the Images and [Replacelcon](#) method to change the Images collection. The Images collection is 1 based. (Currently, the property has no effect). The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection.

property Calc.ImageSize as Long

Retrieves or sets the size of the control' icons.

Type	Description
Long	A long expression that defines the size of icons the control displays.

By default, the ImageSize property is 16 (pixels). The ImageSize property specifies the size of icons being loaded using the [Images](#) method. The control's Images collection is cleared if the ImageSize property is changed, so it is recommended to set the ImageSize property before calling the Images method. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. For instance, if the ICO file to load includes different types the one closest with the size specified by ImageSize property is loaded by Images method. The ImageSize property does NOT change the height for the control's font.

property Calc.Message(Msg as MessageEnum) as String

Retrieves or sets a value that indicates the control's message.

Type	Description
Msg as MessageEnum	A MessageEnum expression that indicates the value being changed
String	A string expression that indicates the message being changed. It supports built-in HTML format.

Use the Message property to customize the string being displayed by the control when an error occurs.

The following sample changes the "Cannot divide by zero.", that's displayed when the calculator performs a division by zero:

```
Calc1.Message(exCannotDivideByZero) = "Divide by zero."
```

The Message property supports built-in HTML format like follows:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... ** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "**bit**" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "**bit**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggbb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggbb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggbb> ... </solidline>** draws a solid-

line on the bottom side of the current text-line, of specified RGB color. The `<solidline> ... </solidline>` draws a black solid-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.

- **`<dotline rrggbb> ... </dotline>` or `<dotline=rrggbb> ... </dotline>`** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The `<dotline> ... </dotline>` draws a black dot-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.
- **`<upline> ... </upline>`** draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).
- **`<r>`** right aligns the text
- **`<c>`** centers the text
- **`
`** forces a line-break
- **`number[:width]`** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **`key[:width]`** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **`&`**; (&), **`<`**; (<), **`>`**; (>), **`"`**; (") and **`&#number;`**; (the character with specified code), For instance, the `€` displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a `#character` and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`
- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `` to define a smaller or a larger font to be displayed. For instance: "Text with `<off 6>`subscript" displays the text such as: Text with subscript The "Text with `<off -6>`superscript" displays the text such as: Text with superscript
- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the `rr/gg/bb` represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `` HTML tag can be used to define the height of the font. Any of the `rrggbb`, mode or

blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<gra FFFFFFFF;1;1>gradient-center</gra>" generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<out 000000><fgcolor=FFFFFF>outlined</fgcolor></out>" generates the following picture:

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<sha>shadow</sha>" generates the following picture:

shadow

or "<sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha>" gets:

outline anti-aliasing

method Calc.Paste ()

Inserts data from the clipboard.

Type	Description
------	-------------

Use the Paste method to paste the control's clipboard in the control's label area. Use the [Caption](#) property to access the control's caption. The [Copy](#) method copies the control's caption to the clipboard.

property Calc.Picture as IPictureDisp

Retrieves or sets a graphic to be displayed in the control.

Type	Description
IPictureDisp	A Picture object that indicates the control's picture.

Use the Picture property to load a picture on the control's background. Use the [PictureDisplay](#) property to arrange the picture on the control's background.

property Calc.PictureDisplay as PictureDisplayEnum

Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates the way how the control's picture is displayed on its background.

Use the [Picture](#) property to load a picture into the control's background. Use the PictureDisplay property to arrange how the control's picture is displayed on its background. Use the [PictureDown](#) and [PictureUp](#) properties to specify the picture for the buttons.

property Calc.PictureDown as Variant

Specifies the picture that's displayed when the button is down.

Type	Description
Variant	A Picture object that indicates the cell's picture. (A Picture object implements IPicture interface), a string expression that indicates the base64 encoded string that holds a picture object. Use the eximages tool to save your picture as base64 encoded format.

Use the PictureDown and [PictureUp](#) properties to specify the picture for the buttons. Use the [Picture](#) property to load a picture into the control's background. Use the [PictureDisplay](#) property to arrange how the control's picture is displayed on its background. Use the [ButtonHeight](#) and [ButtonWidth](#) property to specify the height and the width of the buttons in the control.

The following sample assigns different images  to the buttons:

```
PictureUp =  
"gBHJJGHA5MIgAEIe4AAAFhyFiC9fa9bDbbABjEZjUbjkdj0fkEhkUjkkIk0nIEplUrIktl0vmExmU  
  
PictureDown =  
"gBHJJGHA5MIgAEIe4AAAFhyFiDYiQBikVi0XjEZjUbjkdj0fkEhkUjkkIk0nIEplUrIktl0vmExmUz
```

property Calc.PictureUp as Variant

Specifies the picture that's displayed when the button is up.

Type	Description
Variant	A Picture object that indicates the cell's picture. (A Picture object implements IPicture interface), a string expression that indicates the base64 encoded string that holds a picture object. Use the eximages tool to save your picture as base64 encoded format.

Use the PictureDown and [PictureUp](#) properties to specify the picture for the buttons. Use the [Picture](#) property to load a picture into the control's background. Use the [PictureDisplay](#) property to arrange how the control's picture is displayed on its background. Use the [ButtonHeight](#) and [ButtonWidth](#) property to specify the height and the width of the buttons in the control.

The following sample assigns different images  to the buttons:

```
PictureUp =  
"gBHJJGHA5MIgAEIe4AAAFhyFiC9fa9bDbbABjEZjUbjkdj0fkEhkUjkkIk0nIEplUrIktl0vmExmU  
  
PictureDown =  
"gBHJJGHA5MIgAEIe4AAAFhyFiDYiQBikVi0XjEZjUbjkdj0fkEhkUjkkIk0nIEplUrIktl0vmExmUz
```

method Calc.Refresh ()

Refreshes the control.

Type	Description
	Use the Refresh method to refresh the control. The Buttons property is reinitialized. Call the Refresh method if the user changes the decimal symbol.

method Calc.Replacelcon ([Icon as Variant], [Index as Variant])

Adds a new icon, replaces an icon or clears the control's image list.

Type	Description
Icon as Variant	A long expression that indicates the icon's handle.
Index as Variant	A long expression that indicates the index where icon is inserted.

Return	Description
Long	A long expression that indicates the index of the icon in the images collection.

Use the Replacelcon property to add, remove or replace an icon in the control's images collection. Also, the Replacelcon property can clear the images collection. Use the [Images](#) method to attach an image list to the control. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection.

The following sample shows how to add a new icon to control's images list:

```
i = Calc1.Replacelcon( LoadPicture("d:\icons\help.ico").Handle), in this case the i specifies the index where the icon was added
```

The following sample shows how to replace an icon into control's images list::

```
i = Calc1.Replacelcon( LoadPicture("d:\icons\help.ico").Handle, 0), in this case the i is zero, because the first icon was replaced.
```

The following sample shows how to remove an icon from control's images list:

```
Calc1.Replacelcon 0, i, in this case the i must be the index of the icon that follows to be removed
```

The following sample shows how to clear the control's icons collection:

```
Calc1.Replacelcon 0, -
```

method Calc.Reset ()

Resets the control

Type	Description
------	-------------

Use the Reset method to reset the control. The Reset method erases the control's caption as well as the internal stack of operators and operations. Use the [Caption](#) property to access the control's caption. Use the [Execute](#) method to execute operations.

property Calc.ShowImageList as Boolean

Specifies whether the control's image list window is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the control's images list window is visible or hidden.

The property has effect only at design time. Use the [Images](#) method to attach an image list to the control. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection.

property Calc.Template as String

Specifies the control's template.

Type	Description
String	A string expression that indicates the control's template.

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string (template string). Use the [ExecuteTemplate](#) property to execute a template script and gets the result.

Most of our UI components provide a Template page that's accessible in design mode. No matter what programming language you are using, you can have a quick view of the component's features using the WYSWYG Template editor.

- Place the control to your form or dialog.
- Locate the Properties item, in the control's context menu, in design mode. If your environment doesn't provide a Properties item in the control's context menu, please try to locate in the Properties browser.
- Click it, and locate the Template page.
- Click the Help button. In the left side, you will see the component, in the right side, you will see a x-script code that calls methods and properties of the control.

The control's Template page helps user to initialize the control's look and feel in design mode, using the x-script language that's easy and powerful. The Template page displays the control on the left side of the page. On the right side of the Template page, a simple editor is displayed where user writes the initialization code. The control's look and feel is automatically updated as soon as the user types new instructions. The Template script is saved to the container persistence (when Apply button is pressed), and it is executed when the control is initialized at runtime. Any component that provides a WYSWYG Template page, provides a Template property. The Template property executes code from a string (template string).

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable = property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name*

of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: `h = InsertItem(0,"New Child")`)

- *property(list of arguments) = value Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- *method(list of arguments) Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- *{ Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- *} Ending the object's context*
- *object. property(list of arguments).property(list of arguments).... The .(dot) character splits the object from its property. For instance, the `Columns.Add("Column1").HeaderBackColor = RGB(255,0,0)`, adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with 0x which indicates a hexa decimal representation, else it should starts with digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by # character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by " or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: `BackColor = RGB(255,0,0)`*
- **LoadPicture(file)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property Calc.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus** or **XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be [Template](#) or [ExecuteTemplate](#) property which can use the variable a and b being defined previously.

So, calling the TemplateDef property should be as follows:

```
with (Control)
    TemplateDef = [Dim var_Column]
    TemplateDef = var_Column
    Template = [var_Column.Def(4) = 255]
endwith
```

This sample allocates a variable var_Column, assigns the value to the variable (the second call of the TemplateDef), and the Template call uses the var_Column variable (as an object), to call its Def property with the parameter 4.

Let's say we need to define the background color for a specified column, so we need to call the Def(exCellBackColor) property of the column, to define the color for all cells in the column.

The following **VB6** sample shows setting the Def property such as:

```
With Control
    .Columns.Add("Column 1").Def(exCellBackColor) = 255
    .Columns.Add "Column 2"
    .Items.AddItem 0
    .Items.AddItem 1
```

```
.Items.AddItem 2  
End With
```

In **dBASE Plus**, calling the Def(4) has no effect, instead using the TemplateDef helps you to use properly the Def property as follows:

```
local Control,var_Column  
  
Control = form.ActiveX1.nativeObject  
// Control.Columns.Add("Column 1").Def(4) = 255  
var_Column = Control.Columns.Add("Column 1")  
with (Control)  
    TemplateDef = [Dim var_Column]  
    TemplateDef = var_Column  
    Template = [var_Column.Def(4) = 255]  
endwith  
Control.Columns.Add("Column 2")  
Control.Items.AddItem(0)  
Control.Items.AddItem(1)  
Control.Items.AddItem(2)
```

The equivalent sample for **XBasic in A5**, is as follows:

```
Dim Control as P  
Dim var_Column as P  
  
Control = topparent:CONTROL_ACTIVEX1.activex  
' Control.Columns.Add("Column 1").Def(4) = 255  
var_Column = Control.Columns.Add("Column 1")  
Control.TemplateDef = "Dim var_Column"  
Control.TemplateDef = var_Column  
Control.Template = "var_Column.Def(4) = 255"  
  
Control.Columns.Add("Column 2")  
Control.Items.AddItem(0)  
Control.Items.AddItem(1)  
Control.Items.AddItem(2)
```

The samples just call the `Column.Def(4) = Value`, using the `TemplateDef`. The first call of `TemplateDef` property is `"Dim var_Column"`, which indicates that the next call of the `TemplateDef` will defines the value of the variable `var_Column`, in other words, it defines the object `var_Column`. The last call of the `Template` property uses the `var_Column` member to use the x-script and so to set the `Def` property so a new color is being assigned to the column.

The `TemplateDef`, [Template](#) and [ExecuteTemplate](#) support x-script language (`Template` script of the `Exontrols`), like explained bellow:

The `Template` or x-script is composed by lines of instructions. Instructions are separated by `"\n\r"` (newline characters) or `";"` character. The `;` character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas.* (Sample: `Dim h, h1, h2`)
- `variable = property(list of arguments)` *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas.* (Sample: `h = InsertItem(0,"New Child")`)
- `property(list of arguments) = value` *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- `method(list of arguments)` *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- `{` *Beginning the object's context. The properties or methods called between `{` and `}` are related to the last object returned by the property prior to `{` declaration.*
- `}` *Ending the object's context*
- `object.property(list of arguments).property(list of arguments)....` *The `.` (dot) character splits the object from its property. For instance, the `Columns.Add("Column1").HeaderBackColor = RGB(255,0,0)`, adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may starts with `0x` which indicates a hexa decimal representation, else it should starts with digit, or `+/-` followed by a digit, and `.` is the decimal separator. Sample: `13` indicates the integer `13`, or `12.45` indicates the double expression `12,45`
- *date* expression is delimited by `#` character in the format `#mm/dd/yyyy hh:mm:ss#`. Sample: `#31/12/1971#` indicates the December 31, 1971
- *string* expression is delimited by `"` or ``` characters. If using the ``` character, please

make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(file)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

method Calc.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
NewVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / [TemplateDef](#) property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus** or **XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be [Template](#) or [ExecuteTemplate](#) property which can use the variable a and b being defined previously.

The [TemplateDef](#), TemplatePut, [Template](#) and [ExecuteTemplate](#) support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas.* (Sample: Dim h, h1, h2)
- variable = property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas.* (Sample: h = InsertItem(0,"New Child"))
- property(list of arguments) = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method(list of arguments) *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- { *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- } *Ending the object's context*
- object. property(list of arguments).property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the*

Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.

The x-script may use constant expressions as follows:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may start with 0x which indicates a hexa decimal representation, else it should start with a digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by # character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by " or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also, the template or x-script code may support general functions as follows:

- **Me** property indicates the original object.
- **RGB(R,G,B)** property retrieves an RGB value, where the R, G, B are byte values that indicate the R G B values for the color being specified. For instance, the following code changes the control's background color to red: *BackColor = RGB(255,0,0)*
- **LoadPicture(file)** property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.
- **CreateObject(progID)** property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.

property Calc.ToolTipDelay as Long

Specifies the time in ms that passes before the ToolTip appears.

Type	Description
Long	A long expression that specifies the time in ms that passes before the ToolTip appears.

If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. (Currently, the property has no effect)

property Calc.ToolTipPopDelay as Long

Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

Type	Description
Long	A long expression that specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. Use the [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. (Currently, the property has no effect)

property Calc.Version as String

Retrieves the control's version.

Type	Description
String	A String expression that indicates the control's version.

The version property specifies the control's version.

CalcCombo object

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {63BBFE5C-9D38-48D6-BD06-69CF1AE2F10C}. The object's program identifier is: "ExCalc.CalcCombo". The /COM object module is: "ExCalc.dll"

The Exontrol's CalcCombo component provides calculator features in a drop down version. The CalcCombo object supports properties and methods:

Name	Description
Appearance	Retrieves or sets the control's appearance
BackColor	Specifies the control's background color.
ButtonHeight	Specifies the height of the control's buttons.
Buttons	Specifies the list of buttons in the control.
ButtonWidth	Specifies the width of the control's buttons.
Caption	Specifies the control's caption.
Copy	Copies the control's content to the clipboard.
DropDown	Shows the drop down portion of the control.
DropUp	Hides the drop down portion of the control.
EditBackColor	Specifies the control's edit background color.
EditForeColor	Specifies the control's edit foreground color.
Enabled	Retrieves or sets a value that indicates whether the control is enabled ot disabled.
Execute	Executes a command.
Font	Retrieves or sets the control's font.
ForeColor	Specifies the control's foreground color.
hWnd	Retrieves the control's window handle.
LabelHeight	Specifies the label's height.
Message	Retrieves or sets a value that indicates the control's message.
Paste	Inserts data from the clipboard.
PictureDown	Specifies the picture that's displayed when the button is down.
PictureUp	Specifies the picture that's displayed when the button is up.
Refresh	Refreshes the control.

[Reset](#)

Resets the control

property CalcCombo.Appearance as AppearanceEnum

Retrieves or sets the control's appearance

Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the control's border style.

Use the Appearance property to define the control's border style. Use the Appearance property to hide the control borders.

property CalcCombo.BackColor as Color

Specifies the control's background color.

Type	Description
Color	A color expression that indicates the background color of the control's drop down portion.

Use the BackColor property to specify the background color for the control's drop down portion. Use the [ForeColor](#) property to specify the foreground color for the control's drop down portion. Use the [EditBackColor](#) property to specify the background color of the control's label. Use the [EditForeColor](#) property to specify the foreground color of the control's label.

property CalcCombo.ButtonHeight as Long

Specifies the height of the control's buttons.

Type	Description
Long	A long expression that indicates the height of the buttons, in pixels.

By default, the ButtonHeight property is 24 pixels. Use the ButtonHeight and [ButtonWidth](#) property to specify the height and the width of the buttons in the control. Use the [PictureDown](#) and [PictureUp](#) properties to specify the picture for the buttons. Use the [Buttons](#) property to customize the control's matrix of buttons. Use the [LabelHeight](#) property to specify the height of the control's label.

property CalcCombo.Buttons as String

Specifies the list of buttons in the control.

Type	Description
String	A string expression that indicates the list of buttons being displayed. The rows are separated by chr(13)+chr(10) (vbCrLf) sequence, and the buttons inside the row are separated by ';' character.

The Buttons property specifies the buttons and the layout of the buttons in the control. Use the [PictureDown](#) and [PictureUp](#) properties to specify the picture for the buttons. Use the [ButtonHeight](#) and [ButtonWidth](#) property to specify the height and the width of the buttons in the control. The [ClickButton](#) event occurs when the user clicks a button in the calculator.

Each button supports built-in HTML format like follows:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... ** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "**bit**" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "**bit**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggb> ... </solidline>** draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or **<dotline=rrggb> ... </dotline>** draws a dot-line on

the bottom side of the current text-line, of specified RGB color. The `<dotline> ... </dotline>` draws a black dot-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.

- **`<upline> ... </upline>`** draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).
- **`<r>`** right aligns the text
- **`<c>`** centers the text
- **`
`** forces a line-break
- **`number[:width]`** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **`key[:width]`** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **`&`** glyph characters as **`&`**; (`&`), **`<`**; (`<`), **`>`**; (`>`), **`&qout;`**; (`"`) and **`&#number;`**; (the character with specified code), For instance, the `€` displays the EUR character. The `&` ampersand is only recognized as markup when it is followed by a known letter or a `#` character and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`;
- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `` to define a smaller or a larger font to be displayed. For instance: "Text with `<off 6>`subscript" displays the text such as: Text with subscript The "Text with `<off -6>`superscript" displays the text such as: Text with subscript
- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the `rr/gg/bb` represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `` HTML tag can be used to define the height of the font. Any of the `rrggb`, `mode` or `blend` field may not be specified. The `<gra>` with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "`<gra FFFFFFFF;1;1>`gradient-center`</gra>`" generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<out 000000><fgcolor=FFFFFF>outlined</fgcolor></out>" generates the following picture:

outlined

- **<sha rrggbb;width;offset> ... </sha>** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<sha>shadow</sha>" generates the following picture:

shadow

or "<sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha>" gets:

outline anti-aliasing

By default, the Buttons property is "<fgcolor=0000FF>7</fgcolor>;<fgcolor=0000FF>8</fgcolor>;<fgcolor=0000FF>9</fgcolor>;<fgcolor=FF0000></fgcolor>;C\r\n<fgcolor=0000FF>4</fgcolor>;<fgcolor=0000FF>5</fgcolor>;<fgcolor=0000FF>6</fgcolor>;<fgcolor=FF0000>*</fgcolor>;1\x\r\n<fgcolor=0000FF>1</fgcolor>;<fgcolor=0000FF>2</fgcolor>;<fgcolor=0000FF>3</fgcolor>;<fgcolor=FF0000>-</fgcolor>;sqrt\r\n<fgcolor=0000FF>0</fgcolor>;<fgcolor=0000FF>+/-</fgcolor>;<fgcolor=0000FF>.</fgcolor>;<fgcolor=FF0000>+</fgcolor>;<fgcolor=000080>=</fgcolor>"

Use the [Caption](#) property to access the control's caption. Use the [Execute](#) method to execute operations inside the control.

The following sample adds a new button 'sin' and execute the trigonometric sin function when 'sin' button is clicked:

```
Private Sub Form_Load()
```

```
With CalcCombo1
    .Buttons = .Buttons + vbCrLf + "<b>sin<b>"
End With
End Sub

Private Sub CalcCombo1_ClickButton(ByVal Button As String, Cancel As Variant)
    If (Button = "sin") Then
        With CalcCombo1
            .Execute Sin(.Caption)
        End With
    End If
End Sub
```

property CalcCombo.ButtonWidth as Long

Specifies the width of the control's buttons.

Type	Description
Long	A long expression that indicates the width of the buttons, in pixels.

By default, the ButtonWidth property is 32 pixels. Use the [ButtonHeight](#) and ButtonWidth property to specify the height and the width of the buttons in the control. Use the [PictureDown](#) and [PictureUp](#) properties to specify the picture for the buttons. Use the [Buttons](#) property to customize the control's matrix of buttons.

property CalcCombo.Caption as String

Specifies the control's caption.

Type	Description
String	A string expression that specifies the control's caption.

Use the Caption property to access the control's caption. The [Change](#) event notifies your application when the control's caption is changed. The Caption property erases the control's operators and operation stack and replaces the control's caption. Use the [Execute](#) method to execute operations. Use the [Buttons](#) property to assign a different matrix of digits and operators to the control. By default, the Caption property replaces the control's stack (operators and operations) with giving caption. If the new caption just change the format of the Caption property (includes just HTML tags), the new format is applied to the control's label and control's stack (operators and operations) is not altered. For instance, let's say that the control's label displays the number 78, and during the [Change](#) event your application change the Caption property to "<sha ;;0>" + Caption. In this case, the content of the Caption is not change, instead just the new HTML format is applied to the control's label, so operations on the calculator can continue.

The following sample displays the control's caption as soon as user types characters in the control:

```
Private Sub CalcCombo1_Change()  
    With CalcCombo1  
        Debug.Print .Caption  
    End With  
End Sub
```

The Caption property supports built-in HTML format like follows:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... ** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- ** ... ** displays portions of text with a different font and/or

different size. For instance, the "`bit`" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "`bit`" displays the bit text using the current font, but with a different size.

- **`<fgcolor rrggbb> ... </fgcolor>`** or `<fgcolor=rrgbb> ... </fgcolor>` displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **`<bgcolor rrggbb> ... </bgcolor>`** or `<bgcolor=rrgbb> ... </bgcolor>` displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **`<solidline rrggbb> ... </solidline>`** or `<solidline=rrgbb> ... </solidline>` draws a solid-line on the bottom side of the current text-line, of specified RGB color. The `<solidline> ... </solidline>` draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **`<dotline rrggbb> ... </dotline>`** or `<dotline=rrgbb> ... </dotline>` draws a dot-line on the bottom side of the current text-line, of specified RGB color. The `<dotline> ... </dotline>` draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **`<upline> ... </upline>`** draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).
- **`<r>`** right aligns the text
- **`<c>`** centers the text
- **`
`** forces a line-break
- **`number[:width]`** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **`key[:width]`** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **`&`**; (&), **`<`**; (<), **`>`**; (>), **`"`**; (") and **`&#number;`**; (the character with specified code), For instance, the `€` displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`;
- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset

parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `` to define a smaller or a larger font to be displayed. For instance: "Text with `<off 6>`subscript" displays the text such as: Text with subscript The "Text with `<off -6>`superscript" displays the text such as: Text with subscript

- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `` HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The `<gra>` with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "`<gra FFFFFFFF;1;1>`gradient-center`</gra>`" generates the following picture:

gradient-center

- **`<out rrggbb;width> ... </out>`** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `` HTML tag can be used to define the height of the font. For instance the "`<out 000000><fgcolor=FFFFFF>`outlined`</fgcolor></out>`" generates the following picture:

outlined

- **`<sha rrggbb;width;offset> ... </sha>`** define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `` HTML tag can be used to define the height of the font. For instance the "`<sha>`shadow`</sha>`" generates the following picture:

shadow

or "`<sha 404040;5;0><fgcolor=FFFFFF>`outline anti-aliasing`</fgcolor></sha>`" gets:

outline anti-aliasing

The following sample displays the 'Cannot **execute** the operation' string:

CalcCombo1.Caption = "Cannot **execute** the operation."

method CalcCombo.Copy ()

Copies the control's content to the clipboard.

Type	Description
------	-------------

method CalcCombo.DropDown ()

Shows the drop down portion of the control.

Type	Description
------	-------------

Use the DropDown method to show the drop down portion of the control. Use the [DropUp](#) method to hide the drop down portion of the control.

The following sample shows the drop down portion of the control:

```
Private Sub CalcCombo1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF2) Then
        With CalcCombo1
            .DropDown
        End With
    End If
End Sub
```

method CalcCombo.DropUp ()

Hides the drop down portion of the control.

Type	Description
------	-------------

Use the DropUp method to hide the drop down portion of the control. Use the [DropDown](#) method to show the drop down portion of the control.

The following sample shows the drop down portion of the control:

```
Private Sub CalcCombo1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF2) Then
        With CalcCombo1
            .DropUp
        End With
    End If
End Sub
```

property CalcCombo.EditBackColor as Color

Specifies the control's edit background color.

Type	Description
Color	A color expression that indicates the background color of of the control's label.

Use the EditBackColor property to specify the background color of the control's label. Use the [EditForeColor](#) property to specify the foreground color of the control's label. Use the [ForeColor](#) property to specify the foreground color for the control's drop down portion. Use the [BackColor](#) property to specify the background color for the control's drop down portion.

property CalcCombo.EditForeColor as Color

Specifies the control's edit foreground color.

Type	Description
Color	A color expression that indicates the foreground color of of the control's label.

Use the EditForeColor property to specify the foreground color of the control's label. Use the [EditBackColor](#) property to specify the background color of the control's label. Use the [ForeColor](#) property to specify the foreground color for the control's drop down portion. Use the [BackColor](#) property to specify the background color for the control's drop down portion.

property CalcCombo.Enabled as Boolean

Retrieves or sets a value that indicates whether the control is enabled ot disabled.

Type	Description
Boolean	A boolean expression that indicates whether the control is enabled or disabled.

Use the Enabled property to enable or disable the control.

method CalcCombo.Execute (Command as String)

Executes a command.

Type	Description
Command as String	A string expression that indicates the newly control's caption or the operation being executed.

Use the Execute method to execute commands in the control. The Execute method adds operations and operators to the control's stack. The Execute method does not clear the control's stack. The [Caption](#) property erases the control's operators and operation stack and replaces the control's caption. The [Change](#) event notifies your application when the control's caption is changed.

The following sample multiplies two numbers:

```
With CalcCombo1
    .Execute "12.123"
    .Execute "*"
    .Execute "-123"
    .Execute "="
End With
```

property CalcCombo.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object being used.

Use the Font property to specify the control's font.

property CalcCombo.ForeColor as Color

Specifies the control's foreground color.

Type	Description
Color	A color expression that indicates the foreground color of the control's drop down portion.

Use the ForeColor property to specify the foreground color for the control's drop down portion. Use the [BackColor](#) property to specify the background color for the control's drop down portion. Use the [EditBackColor](#) property to specify the background color of the control's label. Use the [EditForeColor](#) property to specify the foreground color of the control's label.

property CalcCombo.hWnd as Long

Retrieves the control's window handle.

Type	Description
Long	A long value that indicates the handle of the control's window.

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

property CalcCombo.LabelHeight as Long

Specifies the label's height.

Type	Description
Long	A long expression that indicates the height of the control's label.

By default, the LabelHeight property is 21 pixels. Use the LabelHeight property to specify the control's height. The height of the control's drop down portion is automatically computed based on the [ButtonWidth](#), [ButtonHeight](#) and [Buttons](#) properties.

property CalcCombo.Message(Msg as MessageEnum) as String

Retrieves or sets a value that indicates the control's message.

Type	Description
Msg as MessageEnum	A MessageEnum expression that indicates the value being changed
String	A string expression that indicates the message being changed. It supports built-in HTML format.

Use the Message property to customize the string being displayed by the control when an error occurs.

The following sample changes the "Cannot divide by zero.", that's displayed when the calculator performs a division by zero:

```
CalcCombo1.Message(exCannotDivideByZero) = "Divide by zero."
```

The Message property supports built-in HTML format like follows:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... ** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "**bit**" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "**bit**" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or **<fgcolor=rrggbb> ... </fgcolor>** displays text with a specified **foreground** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or **<bgcolor=rrggbb> ... </bgcolor>** displays text with a specified **background** color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or **<solidline=rrggbb> ... </solidline>** draws a solid-

line on the bottom side of the current text-line, of specified RGB color. The `<solidline> ... </solidline>` draws a black solid-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.

- **`<dotline rrggbb> ... </dotline>` or `<dotline=rrggbb> ... </dotline>`** draws a dot-line on the bottom side of the current text-line, of specified RGB color. The `<dotline> ... </dotline>` draws a black dot-line on the bottom side of the current text-line. The `rr/gg/bb` represents the red/green/blue values of the color in hexa values.
- **`<upline> ... </upline>`** draws the line on the top side of the current text-line (requires `<solidline>` or `<dotline>`).
- **`<r>`** right aligns the text
- **`<c>`** centers the text
- **`
`** forces a line-break
- **`number[:width]`** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **`key[:width]`** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **`&`**; (&), **`<`**; (<), **`>`**; (>), **`"`**; (") and **`&#number;`**; (the character with specified code), For instance, the `€` displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a `#character` and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`;
- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `` to define a smaller or a larger font to be displayed. For instance: "Text with `<off 6>`subscript" displays the text such as: Text with subscript The "Text with `<off -6>`superscript" displays the text such as: Text with superscript
- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the `rr/gg/bb` represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `` HTML tag can be used to define the height of the font. Any of the `rrggbb`, mode or

blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<gra FFFFFFFF;1;1>gradient-center</gra>" generates the following picture:

gradient-center

- <out rrggbb;width> ... </out> shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<out 000000><fgcolor=FFFFFF>outlined</fgcolor></out>" generates the following picture:

outlined

- <sha rrggbb;width;offset> ... </sha> define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<sha>shadow</sha>" generates the following picture:

shadow

or "<sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha>" gets:

outline anti-aliasing

method CalcCombo.Paste ()

Inserts data from the clipboard.

Type	Description
------	-------------

property CalcCombo.PictureDown as Variant

Specifies the picture that's displayed when the button is down.

Type	Description
Variant	A Picture object that indicates the cell's picture. (A Picture object implements IPicture interface), a string expression that indicates the base64 encoded string that holds a picture object. Use the eximages tool to save your picture as base64 encoded format.

Use the PictureDown and [PictureUp](#) properties to specify the picture for the buttons.

The following sample assigns different images  to the buttons:

```
PictureUp =  
"gBHJJGHA5MIgAEIe4AAAFhyFiC9fa9bDbbABjEZjUbjkdj0fkEhkUjkkIk0nIEplUrlktl0vmExmU  
  
PictureDown =  
"gBHJJGHA5MIgAEIe4AAAFhyFiDYiQBikVi0XjEZjUbjkdj0fkEhkUjkkIk0nIEplUrlktl0vmExmUz
```

property CalcCombo.PictureUp as Variant

Specifies the picture that's displayed when the button is up.

Type	Description
Variant	A Picture object that indicates the cell's picture. (A Picture object implements IPicture interface), a string expression that indicates the base64 encoded string that holds a picture object. Use the eximages tool to save your picture as base64 encoded format.

Use the [PictureDown](#) and PictureUp properties to specify the picture for the buttons.

The following sample assigns different images  to the buttons:

```
PictureUp =  
"gBHJJGHA5MIgAEIe4AAAFhyFiC9fa9bDbbABjEZjUbjkdj0fkEhkUjkkIk0nIEplUrlktl0vmExmU  
  
PictureDown =  
"gBHJJGHA5MIgAEIe4AAAFhyFiDYiQBikVi0XjEZjUbjkdj0fkEhkUjkkIk0nIEplUrlktl0vmExmUz
```

method CalcCombo.Refresh ()

Refreshes the control.

Type	Description
	Refreshes the control. The Buttons property is reinitialized. Call the Refresh method if the user changes the decimal symbol.

method CalcCombo.Reset ()

Resets the control

Type	Description
------	-------------

CalcCombo events

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {63BBFE5C-9D38-48D6-BD06-69CF1AE2F10C}. The object's program identifier is: "ExCalc.CalcCombo". The /COM object module is: "ExCalc.dll"

The CalcCombo component supports the following events:

Name	Description
Change	Occurs when the control's caption is changed.
ClickButton	Occurs when the user clicks the control's button.
Event	Notifies the application once the control fires an event.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.

event Change ()

Occurs when the control's caption is changed.

Type

Description

The Change event notifies your application when the control's caption is changed. Use the [Caption](#) property to access the control's caption. Use the [Execute](#) method to execute a command. The [KeyDown](#) event occurs when the user presses a key while an object has the focus. By default, the [Caption](#) property replaces the control's stack (operators and operations) with giving caption. If the new caption just change the format of the Caption property (includes just HTML tags), the new format is applied to the control's label and control's stack (operators and operations) is not altered. For instance, let's say that the control's label displays the number 78, and during the Change event your application change the Caption property to "<sha ;;0>" + Caption. In this case, the content of the Caption is not change, instead just the new HTML format is applied to the control's label, so operations on the calculator can continue.

The following sample displays the control's caption as soon as user types characters in the control:

```
Private Sub CalcCombo1_Change()  
    With CalcCombo1  
        Debug.Print .Caption  
    End With  
End Sub
```

event ClickButton (Button as String, Cancel as Variant)

Occurs when the user clicks the control's button.

Type	Description
Button as String	A string expression that indicates the name of the being clicked. The Button parameter does not include the HTML format.
Cancel as Variant	A boolean expression that indicates whether the default operation is executed or canceled.

Use the ClickButton event notifies your application that the user clicks or presses a button. Use the ClickButton event to handle custom operations in the control. Use the [Buttons](#) property to append new buttons to the control. Use the [Execute](#) method to execute operations.

The following VB6 sample adds a new button 'sin' and execute the trigonometric sin function when 'sin' button is clicked:

```
Private Sub Form_Load()  
    With CalcCombo1  
        .Buttons = .Buttons + vbCrLf + "<b>sin<b>"  
    End With  
End Sub
```

```
Private Sub CalcCombo1_ClickButton(ByVal Button As String, Cancel As Variant)  
    If (Button = "sin") Then  
        With CalcCombo1  
            .Execute Sin(.Caption)  
        End With  
    End If  
End Sub
```

The following VB.NET sample adds a new button 'sin' and execute the trigonometric sin function when 'sin' button is clicked:

```
Private Sub Excalc1_ClickButton(ByVal sender As System.Object, ByVal Button As  
System.String, ByRef Cancel As System.Boolean) Handles Excalc1.ClickButton  
    If (Button = "sin") Then  
        With Excalc1
```



```
        .Execute(Math.Sin(.Caption))
    End With
End If
If (Button = "cos") Then
    With Excalc1
        .Execute(Math.Cos(.Caption))
    End With
End If
End Sub
```

The following C# sample adds a new button 'sin' and execute the trigonometric sin function when 'sin' button is clicked:

```
private void excalc1_ClickButton(object sender, string Button, ref bool Cancel)
{
    if (Button == "sin")
        excalc1.Execute(Math.Sin(Convert.ToDouble(excalc1.Caption)).ToString());
    else if (Button == "cos")
        excalc1.Execute(Math.Cos(Convert.ToDouble(excalc1.Caption)).ToString());
}
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

Type	Description
EventID as Long	A Long expression that specifies the identifier of the event. Use the EventParam(-2) to display entire information about fired event (such as name, identifier, and properties).

The Event notification occurs ANY time the control fires an event.

This is useful for X++ language, which does not support event with parameters passed by reference.

In X++ the "Error executing code: FormActiveXControl (data source), method ... called with invalid parameters" occurs when handling events that have parameters passed by reference. Passed by reference, means that in the event handler, you can change the value for that parameter, and so the control will takes the new value, and use it. The X++ is NOT able to handle properly events with parameters by reference, so we have the solution.

The solution is using and handling the Event notification and EventParam method., instead handling the event that gives the "invalid parameters" error executing code.

Let's presume that we need to handle the BarParentChange event to change the _Cancel parameter from false to true, which fires the "Error executing code: FormActiveXControl (data source), method onEvent_BarParentChange called with invalid parameters." We need to know the identifier of the BarParentChange event (each event has an unique identifier and it is static, defined in the control's type library). If you are not familiar with what a type library means just handle the Event of the control as follows:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
    print excalc1.EventParam(-2).toString();
}
```

This code allows you to display the information for each event of the control being fired as in the list bellow:

```
"MouseMove/-606( 1 , 0 , 145 , 36 )" VT_BSTR
"BarParentChange/125( 192998632 , 'B' , 192999592 , =false )" VT_BSTR
"BeforeDrawPart/54( 2 , -1962866148 , =0 , =0 , =0 , =0 , =false )" VT_BSTR
```

```
"AfterDrawPart/55( 2 , -1962866148 , 0 , 0 , 0 , 0 )" VT_BSTR
```

```
"MouseMove/-606( 1 , 0 , 145 , 35 )" VT_BSTR
```

Each line indicates an event, and the following information is provided: the name of the event, its identifier, and the list of parameters being passed to the event. The parameters that starts with = character, indicates a parameter by reference, in other words one that can be changed during the event handler.

Now, we can see that the identifier for the BarParentChange event is 125, so we need to handle the Event event as:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
    ;
    if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem
as HITEM, Cancel as Boolean) */
        excalc1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

The code checks if the BarParentChange (_EventID == 125) event is fired, and changes the third parameter of the event to true. The definition for BarParentChange event can be consulted in the control's documentation or in the ActiveX explorer. So, anytime you need to access the original parameters for the event you should use the EventParam method that allows you to get or set a parameter. If the parameter is not passed by reference, you cannot change the parameter's value.

Now, let's add some code to see a complex sample, so let's say that we need to prevent moving the bar from an item to any disabled item. So, we need to specify the Cancel parameter as not Items.EnableItem(NewItem), in other words cancels if the new parent is disabled. Shortly the code will be:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
    ;
    if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem
as HITEM, Cancel as Boolean) */
        if ( !excalc1.Items().EnableItem( excalc1.EventParam( 2 /*NewItem*/ ) ) )
            excalc1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

In conclusion, anytime the X++ fires the "invalid parameters." while handling an event, you can use and handle the Event notification and EventParam methods of the control

event KeyDown (KeyCode as Integer, Shift as Integer)

Occurs when the user presses a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use KeyDown and [KeyUp](#) event procedures if you need to respond to both the pressing and releasing of a key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
```

In a procedure, you can test for any combination of conditions, as in this example:
If AltDown And CtrlDown Then

The following sample shows the drop down portion of the control:

```
Private Sub CalcCombo1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF2) Then
        With CalcCombo1
            .DropDown
        End With
    End If
End Sub
```

event KeyPress (KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	An integer that returns a standard numeric ANSI keycode

The KeyPress event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the keyascii argument changes the character displayed. Use [KeyDown](#) and [KeyUp](#) event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress does not indicate the physical state of the keyboard; instead, it passes a character. KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

event KeyUp (KeyCode as Integer, Shift as Integer)

Occurs when the user releases a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use the KeyUp event procedure to respond to the releasing of a key.

Calc events

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {63BBFE5C-9D38-48D6-BD06-69CF1AE2F10C}. The object's program identifier is: "ExCalc.CalcCombo". The /COM object module is: "ExCalc.dll"

The Calc component supports the following events:

Name	Description
Change	Occurs when the control's caption is changed.
Click	Occurs when the user presses and then releases the left mouse button over the control.
ClickButton	Occurs when the user clicks the control's button.
DbClick	Occurs when the user dblclk the left mouse button over an object.
Event	Notifies the application once the control fires an event.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
MouseDown	Occurs when the user presses a mouse button.
MouseMove	Occurs when the user moves the mouse.
MouseUp	Occurs when the user releases a mouse button.

event Change ()

Occurs when the control's caption is changed.

Type	Description
------	-------------

The Change event notifies your application when the control's caption is changed. Use the [Caption](#) property to access the control's caption. Use the [Execute](#) method to execute a command. The [KeyDown](#) event occurs when the user presses a key while an object has the focus. By default, the [Caption](#) property replaces the control's stack (operators and operations) with giving caption. If the new caption just change the format of the Caption property (includes just HTML tags), the new format is applied to the control's label and control's stack (operators and operations) is not altered. For instance, let's say that the control's label displays the number 78, and during the Change event your application change the Caption property to "<sha ;;0>" + Caption. In this case, the content of the Caption is not change, instead just the new HTML format is applied to the control's label, so operations on the calculator can continue.

Syntax for Change event, **/NET** version, on:

```
C# private void Change(object sender)
{
}
```

```
VB Private Sub Change(ByVal sender As System.Object) Handles Change
End Sub
```

Syntax for Change event, **/COM** version, on:

```
C# private void Change(object sender, EventArgs e)
{
}
```

```
C++ void OnChange()
{
}
```

```
C++ Builder void __fastcall Change(TObject *Sender)
{
}
```

Delphi

```
procedure Change(ASender: TObject; );  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure Change(sender: System.Object; e: System.EventArgs);  
begin  
end;
```

Power...

```
begin event Change()  
end event Change
```

VB.NET

```
Private Sub Change(ByVal sender As System.Object, ByVal e As System.EventArgs)  
Handles Change  
End Sub
```

VB6

```
Private Sub Change()  
End Sub
```

VBA

```
Private Sub Change()  
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnChange(oCalc)  
RETURN
```

Syntax for Change event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="Change()" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function Change()  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComChange  
    Forward Send OnComChange  
End_Procedure
```

Visual
Objects

```
METHOD OCX_Change() CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_Change()  
{  
}
```

XBasic

```
function Change as v ()  
end function
```

dBASE

```
function nativeObject_Change()  
return
```

The following sample displays the control's caption as soon as user types characters in the control:

```
Private Sub Calc1_Change()  
    With Calc1  
        Debug.Print .Caption  
    End With  
End Sub
```

The following sample displays a different format for the control's label:

```
Private Sub Calc1_Change()  
    With Calc1  
        .Caption = "<sha ;0> <b>" & .Caption  
    End With  
End Sub
```

The following sample displays a different format for the control's label, when negative numbers are displayed

```
Private Sub Calc1_Change()  
    With Calc1
```

```
Dim format As String
format = "<sha ;;0> <b>"
If (.Caption < 0) Then
    format = format & "<fgcolor FF0000>"
End If
.Caption = format & .Caption
End With
End Sub
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the control.

Type

Description

The Click event is fired when the user releases the left mouse button over the control. Use a [MouseDown](#) or [MouseUp](#) event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click event, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the [ClickButton](#) event to notify your application when a button is clicked or pressed.

Syntax for Click event, **/NET** version, on:

```
C# private void Click(object sender)
{
}
```

```
VB Private Sub Click(ByVal sender As System.Object) Handles Click
End Sub
```

Syntax for Click event, **/COM** version, on:

```
C# private void ClickEvent(object sender, EventArgs e)
{
}
```

```
C++ void OnClick()
{
}
```

```
C++ Builder void __fastcall Click(TObject *Sender)
{
}
```

```
Delphi procedure Click(ASender: TObject; );
begin
end;
```

Delphi 8
(.NET
only)

```
procedure ClickEvent(sender: System.Object; e: System.EventArgs);  
begin  
end;
```

Power...

```
begin event Click()  
end event Click
```

VB.NET

```
Private Sub ClickEvent(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles ClickEvent  
End Sub
```

VB6

```
Private Sub Click()  
End Sub
```

VBA

```
Private Sub Click()  
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnClick(oCalc)  
RETURN
```

Syntax for Click event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="Click()" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function Click()  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComClick  
Forward Send OnComClick
```

End_Procedure

Visual
Objects

METHOD OCX_Click() CLASS MainDialog
RETURN NIL

X++

```
void onEvent_Click()
{
}
```

XBasic

```
function Click as v ()
end function
```

dBASE

```
function nativeObject_Click()
return
```

event ClickButton (Button as String, Cancel as Variant)

Occurs when the user clicks the control's button.

Type	Description
Button as String	A string expression that indicates the name of the being clicked. The Button parameter does not include the HTML format.
Cancel as Variant	A boolean expression that indicates whether the default operation is executed or canceled.

Use the ClickButton event notifies your application that the user clicks or presses a button. Use the ClickButton event to handle custom operations in the control. Use the [Buttons](#) property to append new buttons to the control. Use the [Execute](#) method to execute operations.

Syntax for ClickButton event, **/NET** version, on:

C#private void ClickButton(object sender,string Button,ref object Cancel)
{
}

VBPrivate Sub ClickButton(ByVal sender As System.Object,ByVal Button As String,ByRef Cancel As Object) Handles ClickButton
End Sub

Syntax for ClickButton event, **/COM** version, on:

C#private void ClickButton(object sender,
AxEXCALCLib._ICalcComboEvents_ClickButtonEvent e)
{
}

C++void OnClickButton(LPCTSTR Button,VARIANT FAR* Cancel)
{
}

C++ Buildervoid __fastcall ClickButton(TObject *Sender,BSTR Button,Variant * Cancel)
{
}

Delphi procedure ClickButton(ASender: TObject; Button : WideString;var Cancel : OleVariant);
begin
end;

**Delphi 8
(.NET
only)** procedure ClickButton(sender: System.Object; e:
AxEXCALCLib._ICalcComboEvents_ClickButtonEvent);
begin
end;

Powe... begin event ClickButton(string Button,any Cancel)
end event ClickButton

VB.NET Private Sub ClickButton(ByVal sender As System.Object, ByVal e As
AxEXCALCLib._ICalcComboEvents_ClickButtonEvent) Handles ClickButton
End Sub

VB6 Private Sub ClickButton(ByVal Button As String,Cancel As Variant)
End Sub

VBA Private Sub ClickButton(ByVal Button As String,Cancel As Variant)
End Sub

VFP LPARAMETERS Button,Cancel

Xbas... PROCEDURE OnClickButton(oCalc,Button,Cancel)
RETURN

Syntax for ClickButton event, **/COM** version (others), on:

Java... <SCRIPT EVENT="ClickButton(Button,Cancel)" LANGUAGE="JScript">
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">
Function ClickButton(Button,Cancel)
End Function
</SCRIPT>

Visual
Data...

```
Procedure OnComClickButton String llButton Variant llCancel  
    Forward Send OnComClickButton llButton llCancel  
End_Procedure
```

Visual
Objects

```
METHOD OCX_ClickButton(Button,Cancel) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_ClickButton(str _Button,COMVariant /*variant*/ _Cancel)  
{  
}
```

XBasic

```
function ClickButton as v (Button as C,Cancel as A)  
end function
```

dBASE

```
function nativeObject_ClickButton(Button,Cancel)  
return
```

The following sample adds a new button 'sin' and execute the trigonometric sin function when 'sin' button is clicked:

```
Private Sub Form_Load()  
    With Calc1  
        .Buttons = .Buttons + vbCrLf + "<b>sin<b>"  
    End With  
End Sub
```

```
Private Sub Calc1_ClickButton(ByVal Button As String, Cancel As Variant)  
    If (Button = "sin") Then  
        With Calc1  
            .Execute Sin(.Caption)  
        End With  
    End If  
End Sub
```

event DbtClick (Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user dblclk the left mouse button over an object.

Type	Description
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The DbtClick event is fired when the user dbl clicks on the control. Use the DbtClick event to notify your application that a cell has been double-clicked.

Syntax for DbtClick event, **/NET** version, on:

C#

```
private void DbtClick(object sender,short Shift,int X,int Y)
{
}
```

VB

```
Private Sub DbtClick(ByVal sender As System.Object,ByVal Shift As Short,ByVal X
As Integer,ByVal Y As Integer) Handles DbtClick
End Sub
```

Syntax for DbtClick event, **/COM** version, on:

C#

```
private void DbtClick(object sender, AxEXCALCLib._ICalcEvents_DbtClickEvent e)
{
}
```

C++

```
void OnDbtClick(short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall DbtClick(TObject *Sender,short Shift,int X,int Y)
{
```

```
}
```

Delphi

```
procedure DblClick(ASender: TObject; Shift : Smallint;X : Integer;Y : Integer);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure DblClick(sender: System.Object; e:  
AxEXCALCLib._ICalcEvents_DblClickEvent);  
begin  
end;
```

Power...

```
begin event DblClick(integer Shift,long X,long Y)  
end event DblClick
```

VB.NET

```
Private Sub DblClick(ByVal sender As System.Object, ByVal e As  
AxEXCALCLib._ICalcEvents_DblClickEvent) Handles DblClick  
End Sub
```

VB6

```
Private Sub DblClick(Shift As Integer,X As Single,Y As Single)  
End Sub
```

VBA

```
Private Sub DblClick(ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)  
End Sub
```

VFP

```
LPARAMETERS Shift,X,Y
```

Xbase...

```
PROCEDURE OnDblClick(oCalc,Shift,X,Y)  
RETURN
```

Syntax for DblClick event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="DblClick(Shift,X,Y)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function DblClick(Shift,X,Y)  
End Function
```

</SCRIPT>

Visual
Data...

```
Procedure OnComDbClick Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS  
IYY  
    Forward Send OnComDbClick IIShift IIX IYY  
End_Procedure
```

Visual
Objects

```
METHOD OCX_DbClick(Shift,X,Y) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_DbClick(int _Shift,int _X,int _Y)  
{  
}
```

XBasic

```
function DbClick as v (Shift as N,X as OLE::Exontrol.Calc.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.Calc.1::OLE_YPOS_PIXELS)  
end function
```

dBASE

```
function nativeObject_DbClick(Shift,X,Y)  
return
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

Type	Description
EventID as Long	A Long expression that specifies the identifier of the event. Use the EventParam (-2) to display entire information about fired event (such as name, identifier, and properties).

The Event notification occurs ANY time the control fires an event.

Syntax for Event event, **/NET** version, on:

C#	private void Event(object sender,int EventID) { }
VB	Private Sub Event(ByVal sender As System.Object,ByVal EventID As Integer) Handles Event End Sub

Syntax for Event event, **/COM** version, on:

C#	private void Event(object sender, AxEXCALCLib._ICalcComboEvents_EventEvent e) { }
C++	void OnEvent(long EventID) { }
C++ Builder	void __fastcall Event(TObject *Sender,long EventID) { }
Delphi	procedure Event(ASender: TObject; EventID : Integer); begin end;

Delphi 8
(.NET
only)

```
procedure Event(sender: System.Object; e: AxEXCALCLib._ICalcComboEvents_EventEvent);  
begin  
end;
```

Powe...

```
begin event Event(long EventID)  
end event Event
```

VB.NET

```
Private Sub Event(ByVal sender As System.Object, ByVal e As  
AxEXCALCLib._ICalcComboEvents_EventEvent) Handles Event  
End Sub
```

VB6

```
Private Sub Event(ByVal EventID As Long)  
End Sub
```

VBA

```
Private Sub Event(ByVal EventID As Long)  
End Sub
```

VFP

```
LPARAMETERS EventID
```

Xbas...

```
PROCEDURE OnEvent(oCalc,EventID)  
RETURN
```

Syntax for Event event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="Event(EventID)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function Event(EventID)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComEvent Integer lIEventID  
Forward Send OnComEvent lIEventID
```

```
End_Procedure
```

Visual
Objects

```
METHOD OCX_Event(EventID) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_Event(int _EventID)  
{  
}
```

XBasic

```
function Event as v (EventID as N)  
end function
```

dBASE

```
function nativeObject_Event(EventID)  
return
```

This is useful for X++ language, which does not support event with parameters passed by reference.

In X++ the "Error executing code: FormActiveXControl (data source), method ... called with invalid parameters" occurs when handling events that have parameters passed by reference. Passed by reference, means that in the event handler, you can change the value for that parameter, and so the control will take the new value, and use it. The X++ is NOT able to handle properly events with parameters by reference, so we have the solution.

The solution is using and handling the Event notification and EventParam method., instead handling the event that gives the "invalid parameters" error executing code.

Let's presume that we need to handle the BarParentChange event to change the _Cancel parameter from false to true, which fires the "Error executing code: FormActiveXControl (data source), method onEvent_BarParentChange called with invalid parameters." We need to know the identifier of the BarParentChange event (each event has a unique identifier and it is static, defined in the control's type library). If you are not familiar with what a type library means just handle the Event of the control as follows:

```
// Notifies the application once the control fires an event.  
void onEvent_Event(int _EventID)  
{  
    print excalc1.EventParam(-2).toString();  
}
```


This code allows you to display the information for each event of the control being fired as in the list below:

```
"MouseMove/-606( 1 , 0 , 145 , 36 )" VT_BSTR
"BarParentChange/125( 192998632 , 'B' , 192999592 , =false )" VT_BSTR
"BeforeDrawPart/54( 2 , -1962866148 , =0 , =0 , =0 , =0 , =false )" VT_BSTR
"AfterDrawPart/55( 2 , -1962866148 , 0 , 0 , 0 , 0 )" VT_BSTR
"MouseMove/-606( 1 , 0 , 145 , 35 )" VT_BSTR
```

Each line indicates an event, and the following information is provided: the name of the event, its identifier, and the list of parameters being passed to the event. The parameters that starts with = character, indicates a parameter by reference, in other words one that can be changed during the event handler.

Now, we can see that the identifier for the BarParentChange event is 125, so we need to handle the Event event as:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
    ;
    if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem
as HITEM, Cancel as Boolean) */
        excalc1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

The code checks if the BarParentChange (_EventID == 125) event is fired, and changes the third parameter of the event to true. The definition for BarParentChange event can be consulted in the control's documentation or in the ActiveX explorer. So, anytime you need to access the original parameters for the event you should use the EventParam method that allows you to get or set a parameter. If the parameter is not passed by reference, you cannot change the parameter's value.

Now, let's add some code to see a complex sample, so let's say that we need to prevent moving the bar from an item to any disabled item. So, we need to specify the Cancel parameter as not Items.EnableItem(NewItem), in other words cancels if the new parent is disabled. Shortly the code will be:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
```

```
;
if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem
as HITEM, Cancel as Boolean) */
    if ( !excalc1.Items().EnableItem( excalc1.EventParam( 2 /*NewItem*/ ) ) )
        excalc1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

In conclusion, anytime the X++ fires the "invalid parameters." while handling an event, you can use and handle the Event notification and EventParam methods of the control

event KeyDown (KeyCode as Integer, Shift as Integer)

Occurs when the user presses a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use KeyDown and [KeyUp](#) event procedures if you need to respond to both the pressing and releasing of a key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
```

In a procedure, you can test for any combination of conditions, as in this example:
If AltDown And CtrlDown Then

Syntax for KeyDown event, **/NET** version, on:

C#

```
private void KeyDown(object sender,ref short KeyCode,short Shift)
{
}
```

VB

```
Private Sub KeyDown(ByVal sender As System.Object,ByRef KeyCode As Short,ByVal Shift As Short) Handles KeyDown
End Sub
```

Syntax for KeyDown event, **/COM** version, on:

C#

```
private void KeyDownEvent(object sender,
AxEXCALCLib._ICalcComboEvents_KeyDownEvent e)
```

```
{  
}
```

```
C++  
void OnKeyDown(short FAR* KeyCode,short Shift)  
{  
}
```

```
C++  
Builder  
void __fastcall KeyDown(TObject *Sender,short * KeyCode,short Shift)  
{  
}
```

```
Delphi  
procedure KeyDown(ASender: TObject; var KeyCode : Smallint;Shift : Smallint);  
begin  
end;
```

```
Delphi 8  
(.NET  
only)  
procedure KeyDownEvent(sender: System.Object; e:  
AxEXCALCLib._ICalcComboEvents_KeyDownEvent);  
begin  
end;
```

```
Powe...  
begin event KeyDown(integer KeyCode,integer Shift)  
end event KeyDown
```

```
VB.NET  
Private Sub KeyDownEvent(ByVal sender As System.Object, ByVal e As  
AxEXCALCLib._ICalcComboEvents_KeyDownEvent) Handles KeyDownEvent  
End Sub
```

```
VB6  
Private Sub KeyDown(KeyCode As Integer,Shift As Integer)  
End Sub
```

```
VBA  
Private Sub KeyDown(KeyCode As Integer,ByVal Shift As Integer)  
End Sub
```

```
VFP  
LPARAMETERS KeyCode,Shift
```

```
Xbas...  
PROCEDURE OnKeyDown(oCalc,KeyCode,Shift)  
RETURN
```

Syntax for KeyDown event, **/COM** version (others), on:

Java... <SCRIPT EVENT="KeyDown(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">
Function KeyDown(KeyCode,Shift)
End Function
</SCRIPT>

Visual
Data... Procedure OnComKeyDown Short llKeyCode Short llShift
Forward Send OnComKeyDown llKeyCode llShift
End_Procedure

Visual
Objects METHOD OCX_KeyDown(KeyCode,Shift) CLASS MainDialog
RETURN NIL

X++ void onEvent_KeyDown(COMVariant /*short*/ _KeyCode,int _Shift)
{
}

XBasic function KeyDown as v (KeyCode as N,Shift as N)
end function

dBASE function nativeObject_KeyDown(KeyCode,Shift)
return

event **KeyPress** (**KeyAscii** as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	An integer that returns a standard numeric ANSI keycode

The **KeyPress** event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the **keyascii** argument changes the character displayed. Use [KeyDown](#) and [KeyUp](#) event procedures to handle any keystroke not recognized by **KeyPress**, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the **KeyDown** and **KeyUp** events, **KeyPress** does not indicate the physical state of the keyboard; instead, it passes a character. **KeyPress** interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

Syntax for **KeyPress** event, **/NET** version, on:

```
C# private void KeyPress(object sender,ref short KeyAscii)
{
}
```

```
VB Private Sub KeyPress(ByVal sender As System.Object,ByRef KeyAscii As Short)
Handles KeyPress
End Sub
```

Syntax for **KeyPress** event, **/COM** version, on:

```
C# private void KeyPressEvent(object sender,
AxEXCALCLib._ICalcComboEvents_KeyPressEvent e)
{
}
```

```
C++ void OnKeyPress(short FAR* KeyAscii)
{
}
```

```
C++ Builder void __fastcall KeyPress(TObject *Sender,short * KeyAscii)
{
}
```

Delphi

```
procedure KeyPress(ASender: TObject; var KeyAscii : Smallint);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure KeyPressEvent(sender: System.Object; e:  
AxEXCALCLib._ICalcComboEvents_KeyPressEvent);  
begin  
end;
```

Power...

```
begin event KeyPress(integer KeyAscii)  
end event KeyPress
```

VB.NET

```
Private Sub KeyPressEvent(ByVal sender As System.Object, ByVal e As  
AxEXCALCLib._ICalcComboEvents_KeyPressEvent) Handles KeyPressEvent  
End Sub
```

VB6

```
Private Sub KeyPress(KeyAscii As Integer)  
End Sub
```

VBA

```
Private Sub KeyPress(KeyAscii As Integer)  
End Sub
```

VFP

```
LPARAMETERS KeyAscii
```

Xbas...

```
PROCEDURE OnKeyPress(oCalc,KeyAscii)  
RETURN
```

Syntax for KeyPress event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="KeyPress(KeyAscii)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function KeyPress(KeyAscii)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComKeyPress Short Integer KeyAscii  
    Forward Send OnComKeyPress Integer KeyAscii  
End_Procedure
```

Visual
Objects

```
METHOD OCX_KeyPress(KeyAscii) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_KeyPress(COMVariant /*short*/ _KeyAscii)  
{  
}
```

XBasic

```
function KeyPress as v (KeyAscii as N)  
end function
```

dBASE

```
function nativeObject_KeyPress(KeyAscii)  
return
```


event KeyUp (KeyCode as Integer, Shift as Integer)

Occurs when the user releases a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use the KeyUp event procedure to respond to the releasing of a key.

Syntax for KeyUp event, **/NET** version, on:

C#private void KeyUp(object sender,ref short KeyCode,short Shift){}

VBPrivate Sub KeyUp(ByVal sender As System.Object,ByRef KeyCode As Short,ByVal Shift As Short) Handles KeyUpEnd Sub

Syntax for KeyUp event, **/COM** version, on:

C#private void KeyUpEvent(object sender,AxEXCALCLib._ICalcComboEvents_KeyUpEvent e){}

C++void OnKeyUp(short FAR* KeyCode,short Shift){}

C++ Buildervoid __fastcall KeyUp(TObject *Sender,short * KeyCode,short Shift){}

```
}
```

Delphi

```
procedure KeyUp(ASender: TObject; var KeyCode : Smallint;Shift : Smallint);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure KeyUpEvent(sender: System.Object; e:  
AxEXCALCLib._ICalcComboEvents_KeyUpEvent);  
begin  
end;
```

Powe...

```
begin event KeyUp(integer KeyCode,integer Shift)  
end event KeyUp
```

VB.NET

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As  
AxEXCALCLib._ICalcComboEvents_KeyUpEvent) Handles KeyUpEvent  
End Sub
```

VB6

```
Private Sub KeyUp(KeyCode As Integer,Shift As Integer)  
End Sub
```

VBA

```
Private Sub KeyUp(KeyCode As Integer,ByVal Shift As Integer)  
End Sub
```

VFP

```
LPARAMETERS KeyCode,Shift
```

Xbas...

```
PROCEDURE OnKeyUp(oCalc,KeyCode,Shift)  
RETURN
```

Syntax for KeyUp event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="KeyUp(KeyCode,Shift)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function KeyUp(KeyCode,Shift)  
End Function
```

</SCRIPT>

Visual
Data...

```
Procedure OnComKeyUp Short Integer KeyCode Short Integer Shift  
    Forward Send OnComKeyUp Integer KeyCode Integer Shift  
End_Procedure
```

Visual
Objects

```
METHOD OCX_KeyUp(KeyCode,Shift) CLASS MainDialog  
RETURN NIL
```

C++

```
void onEvent_KeyUp(COMVariant /*short*/ _KeyCode,int _Shift)  
{  
}
```

XBasic

```
function KeyUp as v (KeyCode as N,Shift as N)  
end function
```

dBASE

```
function nativeObject_KeyUp(KeyCode,Shift)  
return
```

event MouseDown (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user presses a mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

Use a MouseDown or [MouseUp](#) event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the [Click](#) and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Syntax for MouseDown event, **/NET** version, on:

```
C# private void MouseDownEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

```
VB Private Sub MouseDownEvent(ByVal sender As System.Object,ByVal Button As
Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles
MouseDownEvent
End Sub
```

Syntax for MouseDown event, **/COM** version, on:

```
C# private void MouseDownEvent(object sender,
AxEXCALCLib._ICalcEvents_MouseDownEvent e)
{
}
```

C++ void OnMouseDown(short Button,short Shift,long X,long Y)
{
}

C++ Builder void __fastcall MouseDown(TObject *Sender,short Button,short Shift,int X,int Y)
{
}

Delphi procedure MouseDown(ASender: TObject; Button : Smallint;Shift : Smallint;X : Integer;Y : Integer);
begin
end;

Delphi 8 (.NET only) procedure MouseDownEvent(sender: System.Object; e: AxEXCALCLib._ICalcEvents_MouseDownEvent);
begin
end;

Powe... begin event MouseDown(integer Button,integer Shift,long X,long Y)
end event MouseDown

VB.NET Private Sub MouseDownEvent(ByVal sender As System.Object, ByVal e As AxEXCALCLib._ICalcEvents_MouseDownEvent) Handles MouseDownEvent
End Sub

VB6 Private Sub MouseDown(Button As Integer,Shift As Integer,X As Single,Y As Single)
End Sub

VBA Private Sub MouseDown(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)
End Sub

VFP LPARAMETERS Button,Shift,X,Y

Xbas... PROCEDURE OnMouseDown(oCalc,Button,Shift,X,Y)
RETURN

Syntax for MouseDown event, **/COM** version (others), on:

Java... `<SCRIPT EVENT="MouseDown(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>`

VBSc... `<SCRIPT LANGUAGE="VBScript">
Function MouseDown(Button,Shift,X,Y)
End Function
</SCRIPT>`

Visual Data... `Procedure OnComMouseDown Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
 Forward Send OnComMouseDown IButton IShift IIX IY
End_Procedure`

Visual Objects `METHOD OCX_MouseDown(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL`

X++ `void onEvent_MouseDown(int _Button,int _Shift,int _X,int _Y)
{
}`

XBasic `function MouseDown as v (Button as N,Shift as N,X as
OLE::Exontrol.Calc.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Calc.1::OLE_YPOS_PIXELS)
end function`

dBASE `function nativeObject_MouseDown(Button,Shift,X,Y)
return`

event MouseMove (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user moves the mouse.

Type	Description
Button as Integer	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders. Use the [ButtonFromPoint](#) property to get the button from the cursor.

The following sample displays the button from cursor:

```
Private Sub Calc1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With Calc1
        Debug.Print .ButtonFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)
    End With
End Sub
```

Syntax for MouseMove event, **/NET** version, on:

C#

```
private void MouseMoveEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB

```
Private Sub MouseMoveEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseMoveEvent
End Sub
```

Syntax for MouseMove event, **/COM** version, on:

C#	<pre>private void MouseMoveEvent(object sender, AxEXCALCLib._ICalcEvents_MouseMoveEvent e) { }</pre>
C++	<pre>void OnMouseMove(short Button,short Shift,long X,long Y) { }</pre>
C++ Builder	<pre>void __fastcall MouseMove(TObject *Sender,short Button,short Shift,int X,int Y) { }</pre>
Delphi	<pre>procedure MouseMove(ASender: TObject; Button : Smallint;Shift : Smallint;X : Integer;Y : Integer); begin end;</pre>
Delphi 8 (.NET only)	<pre>procedure MouseMoveEvent(sender: System.Object; e: AxEXCALCLib._ICalcEvents_MouseMoveEvent); begin end;</pre>
Powe...	<pre>begin event MouseMove(integer Button,integer Shift,long X,long Y) end event MouseMove</pre>
VB.NET	<pre>Private Sub MouseMoveEvent(ByVal sender As System.Object, ByVal e As AxEXCALCLib._ICalcEvents_MouseMoveEvent) Handles MouseMoveEvent End Sub</pre>
VB6	<pre>Private Sub MouseMove(Button As Integer,Shift As Integer,X As Single,Y As Single) End Sub</pre>
VBA	<pre>Private Sub MouseMove(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long) End Sub</pre>

VFP

LPARAMETERS Button,Shift,X,Y

Xbas...

```
PROCEDURE OnMouseMove(oCalc,Button,Shift,X,Y)
RETURN
```

Syntax for MouseMove event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="MouseMove(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function MouseMove(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual
Data...

```
Procedure OnComMouseMove Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
    Forward Send OnComMouseMove IButton IShift IIX IY
End_Procedure
```

Visual
Objects

```
METHOD OCX_MouseMove(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_MouseMove(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic

```
function MouseMove as v (Button as N,Shift as N,X as
OLE::Exontrol.Calc.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Calc.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_MouseMove(Button,Shift,X,Y)
return
```

event MouseUp (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user releases a mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

Use a [MouseDown](#) or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the [Click](#) and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Syntax for MouseUp event, **/NET** version, on:

```
C# private void MouseUpEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

```
VB Private Sub MouseUpEvent(ByVal sender As System.Object,ByVal Button As
Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles
MouseUpEvent
End Sub
```

Syntax for MouseUp event, **/COM** version, on:

```
C# private void MouseUpEvent(object sender,
AxEXCALCLib._ICalcEvents_MouseUpEvent e)
{
}
```

C++ void OnMouseUp(short Button,short Shift,long X,long Y)
{
}

C++ Builder void __fastcall MouseUp(TObject *Sender,short Button,short Shift,int X,int Y)
{
}

Delphi procedure MouseUp(ASender: TObject; Button : Smallint;Shift : Smallint;X : Integer;Y : Integer);
begin
end;

Delphi 8 (.NET only) procedure MouseUpEvent(sender: System.Object; e: AxEXCALCLib._ICalcEvents_MouseUpEvent);
begin
end;

Powe... begin event MouseUp(integer Button,integer Shift,long X,long Y)
end event MouseUp

VB.NET Private Sub MouseUpEvent(ByVal sender As System.Object, ByVal e As AxEXCALCLib._ICalcEvents_MouseUpEvent) Handles MouseUpEvent
End Sub

VB6 Private Sub MouseUp(Button As Integer,Shift As Integer,X As Single,Y As Single)
End Sub

VBA Private Sub MouseUp(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)
End Sub

VFP LPARAMETERS Button,Shift,X,Y

Xbas... PROCEDURE OnMouseUp(oCalc,Button,Shift,X,Y)
RETURN

Syntax for MouseUp event, **/COM** version (others), on:

Java...	<SCRIPT EVENT="MouseUp(Button,Shift,X,Y)" LANGUAGE="JScript"> </SCRIPT>
VBSc...	<SCRIPT LANGUAGE="VBScript"> Function MouseUp(Button,Shift,X,Y) End Function </SCRIPT>
Visual Data...	Procedure OnComMouseUp Short lButton Short lShift OLE_XPOS_PIXELS lX OLE_YPOS_PIXELS lY Forward Send OnComMouseUp lButton lShift lX lY End_Procedure
Visual Objects	METHOD OCX_MouseUp(Button,Shift,X,Y) CLASS MainDialog RETURN NIL
X++	void onEvent_MouseUp(int _Button,int _Shift,int _X,int _Y) { }
XBasic	function MouseUp as v (Button as N,Shift as N,X as OLE::Exontrol.Calc.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Calc.1::OLE_YPOS_PIXELS) end function
dBASE	function nativeObject_MouseUp(Button,Shift,X,Y) return